EXHIBIT D

D1

```
/*****************************************************************
 *
 *  This API is an enhancement of the RECOVER_API.
                                        It has been designed to
 *  support restoration of Symmetrix Connect backups as well as
                                                    network
 *  backups.  In addition, it has been implemented in a client/server
 *  architecture, where the restore GUI on an EDM client can browse,
                                                    mark, and
 *  initiate restores from the EDM server.
                                        It uses the Restore Engine API
 *  to access, via ONC RPC,
                    the restore functions provided on the EDM server.
 *
 *  This API is defined to allow access to catalog information without
 *  knowing catalog structures or data types.  This API should remain
 *  extendable through the use of data hiding techniques.
 *
 *  Note:
 *  -----
 *  This API uses the philosophy that the caller is responsible for
                                                    allocation
 *  and freeing of objects.  This should remain consistent throughout.
 *
 *
 *****************************************************************
                                                        ****/
#ifndef H_RESTOREAPI
#define H_RESTOREAPI

#include <eerrno/e_errno.h>
#include <ebcfg#pi/BuCfg_Defs.
                        h>          /* Need this for platform types */

#define RESTORE_API_VERSION    1.0

#define TLO_BITFLAG_NETWORK_RESTORE_NOT_POSSIBLE   0x1
/*
 * Datatypes
 */

/* restorable objects are private */
typedef void *restorableObjectPtr;

/* media objects are private */
typedef void *mediaObjectPtr;

/* feedback objects are private */
typedef void *feedbackObjectPtr;

/* EDMProgress object are private */
typedef void *EDMProgressPtr;

/* WIProgress objects are private */
typedef void *WIProgressPtr;

/* Notify objects are private */
typedef void *NotifyObjectPtr;

/* submit objects are private */
typedef void *submitUserObjectPtr;

/* query objects are opaque to the user */
typedef void *queryObjectPtr ;

1 typedef enum {Backup_Good,
```



```
1                 Backup_Bad,
1                 Backup_Expired,
                  Backup_Child_Without_Data} BackupStatus;

1 typedef enum {Media_Online,
1               Media_Offline,
                Media_Offsite} MediaStatus;

1 typedef enum {Always_Overwrite,
1               Never_Overwrite,
                Older_Only_Overwrite} OverwritePolicy;

1 typedef enum {Files_Only,
1               Directories_Only,
1               Others_Only,
                All_File_Types} FileType;

1 typedef enum {restoreTransportSCSI,
                restoreTransportNetwork} RestoreTransport;

1 typedef enum {EBREC_AllSizes,
1               EBREC_Equal,
1               EBREC_GreaterThan,
1               EBREC_GreaterOrEqual,
1               EBREC_LessThan,
                EBREC_LessOrEqual} MatchType;

  typedef char *hostname_ty;

  typedef void *serverHandle;
                                    /* pointer to CSC handle for Restore Engine
                                                        server */
  #define MAX_TEMPLATE_LEN 64
  typedef char template_name_ty [MAX_TEMPLATE_LEN];

  typedef struct _EDMRST_submit_args
1 {
1   char            *socketClientNm;
1   int             clientSocketPort;
1   char            *mapfile_env;
  }EDMRST_submit_args;




  typedef struct _EBREC_SearchCriteriaRec
1 {
1   char        startDirectory[256]; /* Directory to start searching */
1   boolean_ty descendDirectory;     /* Flag to descend into sub dirs */
1   char        searchString[128];
                                     /* String to search for (as in find) */
1   boolean_ty excludeString;
                                     /* Flag to include or exclude string */
1   FileType    typeOfFile;
                                      /* Types of files to search for */
1   char        owner[64];
                                     /* Specific owner of files (or '\0') */
1   boolean_ty excludeOwner;
                                      /* Flag to include or exclude owner */
1   char        group[64];
                                     /* Specific group of files (or '\0') */
1   boolean_ty excludeGroup;
                                      /* Flag to include or exclude group */
1   u_hyper     sizeInBytes;
                                        /* Specific size of files to find */
1   MatchType   sizeMatch;
                                    /* The type of matching to do for size */
```

```
117 1   time_t      startTime;
                                    /* First backup date to use for search */
118 1   time_t      endTime;
                                    /* Last backup date to use for search */
119     } EBREC_SearchCriteriaRec;

121     /* Definition of bits in flags input to backup time selection
                                                          functions: */
122     #define BACKUP_SELECTION_FLAG_MASK_COMPLETE      0x1
123     #define BACKUP_SELECTION_FLAG_COMPLETE_ONLY      0x1
124     #define BACKUP_SELECTION_FLAG_PARTIAL_OK         0x0


127     #define INIT_COOKIE    0
128     #define DONE_COOKIE   -1


131     /*****************************************************************
                                                                        *
132      * Definitions to be replaced by Restore Engine Header at some point :
133      *****************************************************************
                                                                        */


135 1   typedef enum {
136 1           RE_STATE_TIMEOUT = -4,
137 1           RE_STATE_ADMIN_QUIT = -3,
138 1           RE_STATE_USER_QUIT = -2,
139 1           RE_STATE_FAILED = -1,
140 1           RE_STATE_SUCCESSFUL = 0,
141 1           RE_STATE_STOPPED,
142 1           RE_STATE_RUNNING,
143 1           RE_STATE_STARTING,
144 1           RE_STATE_PREPHASE,
145 1           RE_STATE_POSTPHASE
146     } RERunningState ;

148     /*****************************************************************
149      * Get Backup Servers:
150      *
151      *  This function is provided to allow retrieval of the
152      *  EDM servers which are available to perform restores from.
153      *
154      *  This is a client-side only function.
155      *
156      * Parameters:
157      *   server (
                  O)  - a pointer to a buffer allocated to receive a host name
158      *   cookie (
                 IO) - a place holder for the position in the list of hosts
159      *
160      * NOTE:
161      *       This function will only return a single host name in the 4/99
                                                                    release!
162      *
163      *****************************************************************/

165     eerrno_ty  EDMRST_GetBackupServers(
                                        hostname_ty  server, long *cookie );


168     /*****************************************************************
169      * Initialize:
170      *
171      *  This function initializes a restoral session.  This must be called
172      *  prior to any of the following functions in this API.
173      *
```



```
174      * Parameters:
175      *
176      *  serverName (I) - The machine name of the server to use.
177      *  svrHdl     (
                       O) - A handle to receive a pointer to this user's client
178      *                  handle for the Restore Engine connection.
179      *  timeout    (
                    I) - The maximum number of seconds to wait for the connection
180      *                  to the Restore Engine process to be completed.
181      *  rst_mode   (
                    I) - The mode inwhich a restore is going to be done, either
182      *                  with RAW_NETWORK or PLUGIN.
183      *
184      * Return Codes:
185      *       R_SUCCESS                          - operation completed
                                                          successfully
186      *       EP_RB_RECOVER_BAD_ARGS
187      *       EP_RB_RECOVER_SERVER_FAIL          -If cannot access Dispatch
                                                          daemon host
188      *                                         or restore engine host within
                                                          timeout
189      *       EP_RB_RECOVER_NOMEM
190      *       EP_RB_RECOVER_RPC_FAIL             -If server access error
191      *       EP_RB_RECOVER_PARSE_ERROR          -If unable to use config file
192      *       EP_RB_RECOVER_NO_DIRECTORY         -If unable to find EB (
                                                          sub)directory
193      *       EP_RB_RECOVER_FATALERR             - Internal restore eng error
194      *       EP_RB_RECOVER_PERMISSION_DENIED          -If cannot find user
                                                          in password file
195      *       EP_RB_RECOVER_INVALOP              -If another request executing
196       *****************************************************************/
197     eerrno_ty EDMRST_Initialize ( hostname_ty          serverName,
198                                   serverHandle         *svrHdl,
199                                   unsigned long        timeout);


201     /*****************************************************************
202      * Ping:
203      *
204      *  This function allows a ping to be issued in order to keep the
205      *  engine alive and running so that the engine will not time out.
206      *
207      * Parameters:
208      *
209      *  svrHdl (I) - A pointer to this user's client handle for the
210      *               Restore Engine (server) connection.
211      *
212      *****************************************************************/
213     eerrno_ty EDMRST_Ping( serverHandle svrHdl );


216     /*****************************************************************
217      * Finish:
218      *
219      *  This function terminates a restoral session,
                                                but only during the browse and
220      *  mark phase.
                It will be rejected if a restore is currently being executed.
221      *  This routine will clean up any local memory used in the session
                                                                    and will
222      *  disconnect from the Restore Engine.  After calling this function,
223      *  EDMRST_Initialize MUST be called before calling any other
                                                          functions in this
224      *  API.
225      *
226      * Parameters:
227      *
```

```
 *  svrHdl (I) - A pointer to this user's client handle for the
 *                  Restore Engine (server) connection.
 *
 ************************************************************************/
eerrno_ty EDMRST_Finish( serverHandle svrHdl );


/************************************************************************/
/* Storage Allocation/Deallocation Functions                            */
/************************************************************************/


/*********************************************************************
 * Alloc Restorable Objects:
 *
 *  This routine allocates space for the given number of restorable
 *  objects. The caller MUST pass an array of restorable objects.
 *
 * Parameters:
 *  svrHdl              (
 *                  I) - A pointer to this user's client handle for the
 *                           Restore Engine (server) connection. (
 *                                                       must be valid)
 *  restorableObjects (O) - The array of objects to allocate
 *  count             (I) - The number of objects to allocate.
 *
 * Return Codes:
 *       E_SUCCESS                  - operation completed successfully
 *       EP_RB_RECOVER_BAD_ARGS
 *       EP_RB_RECOVER_NOMEM
 *********************************************************************/
eerrno_ty  EDMRST_AllocRestorableObjects( serverHandle          svrHdl,
                                          restorableObjectPtr
                                                        *restorableObjs,
                                          const short           count );

/*********************************************************************
 * Free Restorable Objects:
 *
 *  This routine frees the space for the given number of restorable
 *  objects. The caller MUST pass an array of restorable objects.
 *
 * Parameters:
 *  svrHdl              (
 *                  I) - A pointer to this user's client handle for the
 *                           Restore Engine (server) connection. (
 *                                                       must be valid)
 *  restorableObjects (IO) - The array of objects to free
 *  count             (I) - The number of objects to free.
 *
 * Return Codes:
 *       E_SUCCESS                  - operation completed successfully
 *       EP_RB_RECOVER_BAD_ARGS
 *       EP_RB_RECOVER_INVALOP    - input objType is invalid
 *********************************************************************/
eerrno_ty  EDMRST_FreeRestorableObjects( serverHandle          svrHdl,
                                         restorableObjectPtr
                                                        *restorableObjects,
                                         const short           count);


/*********************************************************************
 * CopyRestorableObject:
 *
 *  This routine copies a restorable object allocating dynamic field.
```



```
 *
 * Parameters:
 *  svrHdl              (
 *                  I) - A pointer to this user's client handle for the
 *                           Restore Engine (server) connection. (
 *                                                       must be valid)
 *  sourceObject      (I) - The restorable object to copy from.
 *  destinationObject (
 *                  O) - The pre-allocated restorable object to copy to.
 *
 ************************************************************************/
eerrno_ty EDMRST_CopyRestorableObject( serverHandle          svrHdl,
                                       restorableObjectPtr
                                                        sourceObject,
                                       restorableObjectPtr
                                                   *destinationObject );



/*********************************************************************
 * Alloc Media Objects:
 *
 *  This routine allocates space for the given number of media
 *  objects. The caller MUST pass an array of media objects.
 *
 * Parameters:
 *  svrHdl        (I) - A pointer to this user's client handle for the
 *                        Restore Engine (server) connection. (
 *                                                       must be valid)
 *  mediaObjects (O) - The array of objects to allocate
 *  count         (I) - The number of objects to allocate.
 *
 * Return Codes:
 *       E_SUCCESS                  - operation completed successfully
 *       EP_RB_RECOVER_BAD_ARGS
 *       EP_RB_RECOVER_NOMEM
 *********************************************************************/
eerrno_ty  EDMRST_AllocMediaObjects( serverHandle   svrHdl,
                                     mediaObjectPtr *mediaObjects,
                                     const short    count );

/*********************************************************************
 * Free Media Objects:
 *
 *  This routine frees the space for the given number of media
 *  objects. The caller MUST pass an array of media objects.
 *
 * Parameters:
 *  svrHdl        (
 *                I) - A pointer to this user's client handle for the
 *                        Restore Engine (server) connection. (
 *                                                       must be valid)
 *  mediaObjects (IO) - The array of objects to free
 *  count         (I) - The number of objects to free.
 *
 * Return Codes:
 *       E_SUCCESS                  - operation completed successfully
 *       EP_RB_RECOVER_BAD_ARGS
 *       EP_RB_RECOVER_INVALOP    - input objType is invalid
 *********************************************************************/
eerrno_ty  EDMRST_FreeMediaObjects( serverHandle   svrHdl,
                                    mediaObjectPtr *mediaObjects,
                                    const short    count );

/*********************************************************************
 * Alloc Query Object:
 *
```

```
345     *  This routine allocates space for one query object.
346     *
347     * Parameters:
348     *  svrHdl  (I) - A pointer to this user's client handle for the
349     *                Restore Engine (server) connection. (must be valid)
350     *  queryObject (O) - Pointer to the query object
351     *
352     * Return Codes:
353     *       E_SUCCESS                - operation completed successfully
354     *       EP_RB_RECOVER_BAD_ARGS
355     *       EP_RB_RECOVER_NOMEM
356     ************************************************************/
357    eerrno_ty  EDMRST_AllocQueryObject( serverHandle   svrHdl,
358                                        queryObjectPtr *queryObject );
359
360    /*****************************************************************
361     * Free Query Object:
362     *
363     *  This routine frees the space for one query object.
364     *
365     * Parameters:
366     *  svrHdl      (I)  - A pointer to this user's client handle for the
367     *                     Restore Engine (server) connection. (
368                                                          must be valid)
368     *  queryObject (
369                IO) - Ptr to the query object to free -- will be set to NULL
370     *
371     * Return Codes:
372     *       E_SUCCESS                - operation completed successfully
373     *       EP_RB_RECOVER_BAD_ARGS
374     *       EP_RB_RECOVER_INVALOP    - input objType is invalid
375     ************************************************************/
376    eerrno_ty  EDMRST_FreeQueryObject( serverHandle   svrHdl,
377                                       queryObjectPtr *queryObject );
378
379    /*****************************************************************
380     * Alloc Feedback Object:
381     *
382     *  This routine allocates space for one feedback object.
383     *
384     * Parameters:
385     *  svrHdl  (I) - A pointer to this user's client handle for the
386     *                Restore Engine (server) connection. (Not used)
387     *  feedbackObject (O) - The pointer to the allocated object
388     *
389     * Return Codes:
390     *       E_SUCCESS                - operation completed successfully
391     *       EP_RB_RECOVER_BAD_ARGS
392     *       EP_RB_RECOVER_NOMEM
393     ************************************************************/
394    eerrno_ty  EDMRST_AllocFeedbackObject( serverHandle   svrHdl,
395                                           feedbackObjectPtr
396                                                   *feedbackObject );
397
398    /*****************************************************************
399     * Free Feedback Object:
400     *
401     *  This routine frees the space for one feedback object.
402     *
403     * Parameters:
404     *  svrHdl        (
405                      I)  - A pointer to this user's client handle for the
406     *                      Restore Engine (server) connection. (
407                                                          Not used)
408     *  feedbackObjectPtr (
409                    IO) - The feedback object to free -- will be set to NULL
```



```
410     *
411     * Return Codes:
412     *       E_SUCCESS                - operation completed successfully
413     *       EP_RB_RECOVER_BAD_ARGS
414     *       EP_RB_RECOVER_NOMEM
415     ************************************************************/
416    eerrno_ty  EDMRST_FreeFeedbackObject( serverHandle      svrHdl,
417                                          feedbackObjectPtr
418                                                  *feedbackObject );
419
420    /*****************************************************************
421     * EDMRST_GetFeedbackStatus
422     *
423     * Function Description:
424     *       Returns the pointer to the string that contains the text
425     *       for the provided enum.
426     *
427     * Parameters:
428     *    status (
429     *         I) This is the enum that is returned from a GetFeedbackObject
430     *                  call to the restore engine
431     *
432     * Return:
433     *       On success: a pointer to a NULL-terminated string which is the
434     *                   text for the provided enum;
435     *       On failure: a NULL pointer
436     ************************************************************/
437    char * EDMRST_GetFeedbackStatus(RERunningState status);
438
439    /*****************************************************************/
440    /* The following functions obtain configuration information      */
441
442    /* from the eb.cfg database                                      */
443    /*****************************************************************/
444
445    /*****************************************************************
446     * Get Source Hosts:
447     *
448     *  This function is provided to allow retrieval of the
449     *  hosts which are restorable by a given user.
450     *
451     *  Goal:
452     *    For a host to be restorable there must have been at least one
453     *    successful backup.
454     *
455     *  The cookie must be initialize to INIT_COOKIE on the first call to
456                                                                    this
456     *  routine.
457     *          This routine will update the cookie to allow retrieval of more
458     *  objects if there is more than "maxEntries".  The cookie will be
459     *  returned as DONE_COOKIE when there are no more to retrieve.
460     *
461     * Parameters:
462     *  svrHdl          (
463     *                   I)  - A pointer to this user's client handle for the
464     *                         Restore Engine (server) connection.
465     *  hostname        (I)  - If NULL, its a no-op. Otherwise, the list of
466     *                         recoverable hosts will be filtered based on
467     *                         the value of "hostname".
468     *  maxEntries      (I)  - the maximum number of hosts to return
469     *  hosts           (O)  - a pre-allocated array to return the hosts in
470     *  numberEntries (
471     *                 O)  - the real number of hosts returned in the array
472     *  cookie          (IO) - a place holder for the list position
473     *
474     ************************************************************/
```

```
eerrno_ty  EDMRST_GetSourceHosts( serverHandle    svrHdl,
                                  const char      *hostname,
                                  const short     maxEntries,
                                  hostname_ty     *hosts,
                                  short           *numberEntries,
                                  long            *cookie );



/*****************************************************************
 * Get Destination Hosts:
 *
 *   This function is provided to allow retrieval of the
 *   hosts which are allowable destinations for the source host
 *   by a given user.
 *
 *   The cookie must be initialize to INIT_COOKIE on the first call to
                                                              this
 *   routine.
        This routine will update the cookie to allow retrieval of more
 *   objects if there is more than "maxEntries".  The cookie will be
 *   returned as DONE_COOKIE when there are no more to retrieve.
 *
 * Parameters:
 *   svrHdl          (
                  I)  - A pointer to this user's client handle for the
 *                        Restore Engine (server) connection.
 *   maxEntries     (I)  - the maximum number of hosts to return
 *   hosts          (O)  - a pre-allocated array to return the hosts in
 *   numberEntries  (
                    O)  - the real number of hosts returned in the array
 *   cookie         (IO) - a place holder for the list position
 *
 *****************************************************************/
eerrno_ty  EDMRST_GetDestinationHosts( serverHandle       svrHdl,
                                       const short        maxEntries,
                                       hostname_ty        *hosts,
                                       short
                                                          *numberEntries,
                                       long               *cookie );



/*****************************************************************
 * Get Top Level Objects (formerly Get Work Items)
 *
 *   This function is provided to allow retrieval of the  work items (
                                                                  for
 *   network backups) and work item sets (
                                  for Symmetrix Connect backups),
 *   which are restorable for the given client.
 *
 *   It is a GOAL of this routine to return all objects ever backed
 *   up successfully.  Currently, though, it only looks in the config
 *   file for 'top level objects' of the given client.
 *
 *   The cookie must be initialize to INIT_COOKIE on the first call to
                                                              this
 *   routine.
        This routine will update the cookie to allow retrieval of more
 *   objects if there is more than "maxEntries".  The cookie will be
 *   returned as DONE_COOKIE when there are no more to retrieve.
 *
 * Parameters:
 *   svrHdl          (
                  I)  - A pointer to this user's client handle for the
 *                        Restore Engine (server) connection.
 *   sourceHost     (
```



```
                  I)  - the name of the host whose backups are being restored
 *   maxEntries     (I)  - the maximum number of objects to return
 *   topLevObjs     (
                    O)  - a pre-allocated array to return the objects in
 *   numberEntries  (
                    O)  - the real number of objects returned in the array
 *   cookie         (IO) - a place holder for the list position
 *
 *****************************************************************/
eerrno_ty  EDMRST_GetTopLevelObjects( serverHandle         svrHdl,
                                      const char           *sourceHost,
                                      const short          maxEntries,
                                      restorableObjectPtr  *workItems,
                                      short
                                                           *numberEntries,
                                      long                 *cookie );



/*****************************************************************
 * Get All Top Level Objects (formerly Get Work Items)
 *
 *   This function is provided to allow retrieval of the  work items (
                                                                  for
 *   network backups) and work item sets (
                                  for Symmetrix Connect backups),
 *   which are restorable for the given client.
 *
 *   It is a GOAL of this routine to return all objects ever backed
 *   up successfully.  Currently, though, it only looks in the config
 *   file for 'top level objects' of the given client.
 *
 *   The cookie must be initialize to INIT_COOKIE on the first call to
                                                              this
 *   routine.
        This routine will update the cookie to allow retrieval of more
 *   objects if there is more than "maxEntries".  The cookie will be
 *   returned as DONE_COOKIE when there are no more to retrieve.
 *
 * Parameters:
 *   svrHdl          (
                  I)  - A pointer to this user's client handle for the
 *                        Restore Engine (server) connection.
 *   sourceHost     (
                  I)  - the name of the host whose backups are being restored
 *   maxEntries     (I)  - the maximum number of objects to return
 *   topLevObjs     (
                    O)  - a pre-allocated array to return the objects in
 *   numberEntries  (
                    O)  - the real number of objects returned in the array
 *   cookie         (IO) - a place holder for the list position
 *
 *****************************************************************/
eerrno_ty  EDMRST_GetAllTopLevelObjects( serverHandle      svrHdl,
                                         const char        *sourceHost,
                                         const short       maxEntries,
                                         restorableObjectPtr *workItems,
                                         short
                                                           *numberEntries,
                                         long              *cookie );



/*****************************************************************
 * Get Top Level Object Templates:
 *
 *   This function is provided to allow retrieval of the
 *   templates to which a work item (top level object) belongs.
 *
```

```
573     *  This routine is provided in the event that the goal of the
574     *  work-item time routines to include all templates cannot be met.
575     *
576     *  The cookie must be initialize to INIT_COOKIE on the first call to
                                                                        this
577     *  routine.
            This routine will update the cookie to allow retrieval of more
578     *  objects if there is more than "maxEntries".  The cookie will be
579     *  returned as DONE_COOKIE when there are no more to retrieve.
580     *
581     *  Parameters:
582     *   svrHdl          (
                  I)   - A pointer to this user's client handle for the
583     *                         Restore Engine (server) connection.
584     *   topLevelObj    (I)   - the top Level object in question
585     *   maxEntries     (I)   - the maximum number of templates to return
586     *   templates      (
                        O)   - a pre-allocated array to return the templates in
587     *   numberEntries (
                     O)   - the real number of templates returned in the array
588     *   cookie          (IO) - a place holder for the list position
589     *
590     ****************************************************************/
591    eerrno_ty  EDMRST_GetTopLevelTtemplates( serverHandle      svrHdl,
592                                             const restorableObjectPtr
                                                                    topLevelObj,
593                                             const short       maxEntries,
594                                             template_name_ty *templates,
595                                             short
                                                              *numberEntries,
596                                             long              *cookie );

598    /*****************************************************************
599     * Does Alternate Exist:
600     *
601     *  This routine determines if an alternate trailset exists for the
                                                                        given
602     *  template.
603     *
604     *  Parameters:
605     *   svrHdl        (I) - A pointer to this user's client handle for the
606     *                       Restore Engine (server) connection.
607     *   topLevelObj   (I) - the top Level object in question
608     *   template      (I) - The name of the template to look for
609     *   exists        (
                     O) - Return flag for whether or not the alternate exists
610     *   svrHdl        (I) - A pointer to this user's client handle for the
611     *                       Restore Engine (server) connection.
612     *   topLevelObj   (I) - the top Level object in question
613     *   template      (I) - The name of the template to look for
614     *   exists        (
                     O) - Return flag for whether or not the alternate exists
615     *
616     ****************************************************************/
617    eerrno_ty EDMRST_DoesAlternateExist( serverHandle             svrHdl,
618                                         const restorableObjectPtr
                                                                    topLevelObj,
619                                         const template_name_ty
                                                                 template_name,
620                                         boolean_ty              *exists );

622    /*****************************************************************
623     * Set Top Level (formerly Work Item) Template:
624     *
625     *  This routine sets the template to be used by the specified top
                                                                        level
```



```
626     *  object (work item, or CBO) and specifies whether or not to use the
627     *  alternate trailset.
628     *
629     *  Parameters:
630     *  svrHdl     (I) - A pointer to this user's client handle for the
631     *                       Restore Engine (server) connection.
632     *  workItem   (I) - The top level object to update
633     *  template   (I) - The name of the template to use
634     *  alternate (I) - Flag whether or not to use the alternate trailset
635     *
636     ****************************************************************/
637    eerrno_ty EDMRST_SetTopLevelTemplate( serverHandle             svrHdl,
638                                          restorableObjectPtr     workItem,
639                                          const template_name_ty
                                                                 template_name,
640                                          const boolean_ty       alternate
                                                                          );

643    /*****************************************************************
                                                                        **/
644    /* Backup Configuration Query Functions                            */
645    /*****************************************************************
                                                                        **/

647    /*****************************************************************
648     * Get Current Template:
649     *
650     *  This routine returns the name of the template that is used by
651     *       the currently selected top level object (
                                                        work item) and the flag
652     *       on whether or not the alternate trail is being used.
653     *
654     *  Parameters:
655     *  svrHdl     (I) - A pointer to this user's client handle for the
656     *                       Restore Engine (server) connection.
657     *  template  (O) - The name of the current template
658     *  alternate (
                  O) - Flag whether or not the alternate trailset is being used.
659     *
660     ****************************************************************/
661    eerrno_ty EDMRST_GetCurrentTemplate( serverHandle      svrHdl,
662                                         template_name_ty template_name,
663                                         boolean_ty        *alternate );

666    /*****************************************************************
667     * GetCurrentBackupTime:
668     *
669     *  This routine returns the time of the backup as selected for the
670     *  work item.
671     *
672     *  Parameters:
673     *  svrHdl (I) - A pointer to this user's client handle for the
674     *                       Restore Engine (server) connection.
675     *  time   (O) - The time the backup for the work-item was run
676     *
677     ****************************************************************/
678    eerrno_ty EDMRST_GetCurrentBackupTime( serverHandle svrHdl,
679                                           time_t        *time );

681    /*****************************************************************/
682    /* Backup Time Selection Functions                                 */
683    /*****************************************************************/
```

```
685
686   /*****************************************************************
687    * SetPrevBackup
688    *
689    *  This routine sets the recover environment to the previous backup
690    *  of the currently selected work item
691    *
692    *  Goal (?):
693    *    This includes both primary and alternate trailsets for all
694    *    templates for which this work-item has backups.
695    *
696    * Parameters:
697    *  svrHdl (I) - A pointer to this user's client handle for the
698    *               Restore Engine (server) connection.
699    *  flags  (
700    *          I) - Selection Flags: e.g., Complete backups only/partial ok
701    *
702    *****************************************************************/
703   eerrno_ty  EDMRST_SetPrevBackup( serverHandle    svrHdl,
704                                    u_long          flags );
705
706   /*****************************************************************
707    * SetNextBackup
708    *
709    *  This routine sets the recover environment to the the next backup
710    *  of the specified work item.
711    *
712    *  Goal:
713    *    This includes both primary and alternate trailsets for all
714    *    templates for which this work-item has backups.
715    *
716    * Parameters:
717    *  svrHdl (I) - A pointer to this user's client handle for the
718    *               Restore Engine (server) connection.
719    *  flags  (
720    *          I) - Selection Flags: e.g., Complete backups only/partial ok
721    *
722    *****************************************************************/
723   eerrno_ty  EDMRST_SetNextBackup( serverHandle    svrHdl,
724                                    u_long          flags );
725
726   /*****************************************************************
727    * SetFirstBackup
728    *
729    *  This routine sets the recover environment to the first backup
730    *  kept for the specified work item.
731    *
732    *  Goal:
733    *    This includes both primary and alternate trailsets for all
734    *    templates for which this work-item has backups.
735    *
736    * Parameters:
737    *  svrHdl (I) - A pointer to this user's client handle for the
738    *               Restore Engine (server) connection.
739    *  flags  (
740    *          I) - Selection Flags: e.g., Complete backups only/partial ok
741    *
742    *****************************************************************/
743   eerrno_ty  EDMRST_SetFirstBackup( serverHandle    svrHdl,
744                                     u_long          flags );
745
746   /*****************************************************************
747    * SetMostRecentBackup
748    *
749    *  This routine sets the recover environment to the most recent
                                                                backup
```



```
746    *  of the specified work item.
747    *
748    *  Goal:
749    *    This includes both primary and alternate trailsets for all
750    *    templates for which this work-item has backups.
751    *
752    * Parameters:
753    *  svrHdl (I) - A pointer to this user's client handle for the
754    *               Restore Engine (server) connection.
755    *  flags  (
756    *          I) - Selection Flags: e.g., Complete backups only/partial ok
757    *
758    *****************************************************************/
759   eerrno_ty  EDMRST_SetMostRecentBackup(   serverHandle     svrHdl,
760                                            u_long           flags );
761
762   /*****************************************************************
763    * Get All Backup Times:
764    *
765    *  This function is provided to allow retrieval of the times which
766    *  the given work-item was backed up.
767    *
768    *  The cookie must be initialize to INIT_COOKIE on the first call to
                                                                this
769    *  routine.
770    *       This routine will update the cookie to allow retrieval of more
771    *  objects if there is more than 'maxEntries'.  The cookie will be
772    *  returned as DONE_COOKIE when there are no more to retrieve.
773    *
774    * Parameters:
775    *  svrHdl          (
776    *                   I)  - A pointer to this user's client handle for the
777    *                                 Restore Engine (server) connection.
778    *  startTime       (I)  - first time which should be returned (
779                                                        nothing before)
780    *  endTime         (I)  - last time which should be returned (
781                                                        nothing after)
782    *  flags           (
783    *                   I)  - Backup constraint flags: e.g. full-only/partial-ok
784    *  maxEntries      (I)  - the maximum number of templates to return
785    *  times           (O)  - a pre-allocated array to return the times in
786    *  numberEntries   (
787    *                   O)  - the real number of templates returned in the array
788    *  cookie          (IO) - a place holder for the list position
789    *
790    *****************************************************************/
791   eerrno_ty  EDMRST_GetAllBackupTimes( serverHandle svrHdl,
792                                        const time_t startTime,
793                                        const time_t endTime,
794                                         u_long      flags,
795                                        const short maxEntries,
796                                        time_t *times,
797                                        short *numberEntries,
798                                        long *cookie );
799
800   /*****************************************************************
801    * Set Backup for Time
802    *
803    *  This routine sets the recover environment to the given backup time
804    *
805    *  Goal:
806    *    This includes both primary and alternate trailsets for all
807    *    templates for which this work-item has backups.
808    *
809    * Parameters:
810    *  svrHdl (I) - A pointer to this user's client handle for the
```

```
 *                   Restore Engine (server) connection.
 *  time   (I) - The time desired
 *  flags  (
         I) - Selection Flags: e.g., Complete backups only/partial ok
 *
 ********************************************************************/
eerrno_ty  EDMRST_SetBackupForTime( serverHandle svrHdl,
                                    const time_t time,
                                    u_long       flags );


/*********************************************************************
 * Get Restorable Objects:
 *
 *  These functions are provided to allow retrieval of the
 *  child objects which are restorable given the parent object. The
 *  caller specifies the parent object and whether or not
 *  to include bad files.
 *
 *  The cookie must be initialize to INIT_COOKIE on the first call to
                                                                  this
 *  routine.
        This routine will update the cookie to allow retrieval of more
 *  objects if there is more than "maxEntries".  The cookie will be
 *  returned as DONE_COOKIE when there are no more to retrieve.
 *
 * Parameters:
 *   svrHdl          (
                 I)  - A pointer to this user's client handle for the
 *                          Restore Engine (server) connection.
 *   parentObject   (I)  - the parent object
 *   allowBadFiles  (I)  - flag whether or not to include bad files
 *   maxEntries     (I)  - the maximum number of objects to return
 *   objects        (
                   O)  - a pre-allocated array to return the objects in
 *   numberEntries  (
                 O)  - the real number of objects returned in the array
 *   cookie         (IO) - a place holder for the list position
 *
 ********************************************************************/
eerrno_ty  EDMRST_GetRestorableObjects( serverHandle        svrHdl,
                                        const restorableObjectPtr
                                                       parentObject,
                                       const boolean_ty
                                                      allowBadFiles,
                                       const long        maxEntries,
                                       restorableObjectPtr *objects,
                                       long
                                                     *numberEntries,
                                       long               *cookie );


/*********************************************************************
 * Restorable Object Access Routines:
 *
 *  These routines retrieve information pertinent to a given
                                                 restorable object.
 *
 * Notes:
 *
 *  GetObjectSize      - returns the size of the individual object
 *
 * Parameters:
 *
 *  svrHdl     (I) - A pointer to this user's client handle for the
 *                   Restore Engine (server) connection.
```



```
 *  thisObject (I) - The restoral object
 *
 *********************************************************************/

const char *EDMRST_GetObjectBaseName( serverHandle        svrHdl,
                                      restorableObjectPtr thisObject
                                                                   );

const char *EDMRST_GetObjectFullName( serverHandle        svrHdl,
                                      restorableObjectPtr thisObject
                                                                   );

boolean_ty EDMRST_IsObjectTopLevel( serverHandle        svrHdl,
                                    restorableObjectPtr thisObject );

boolean_ty EDMRST_IsObjectContainer( serverHandle        svrHdl,
                                     restorableObjectPtr thisObject );

boolean_ty EDMRST_IsObjectLeaf( serverHandle        svrHdl,
                                restorableObjectPtr thisObject );

const char *EDMRST_GetObjectTypeString( serverHandle        svrHdl,
                                        restorableObjectPtr
                                                        thisObject );

u_long EDMRST_GetObjectPermissions( serverHandle        svrHdl,
                                    restorableObjectPtr thisObject );

const char *EDMRST_GetObjectOwnerString( serverHandle        svrHdl,
                                         restorableObjectPtr
                                                        thisObject );

const char *EDMRST_GetObjectGroupString( serverHandle        svrHdl,
                                         restorableObjectPtr
                                                        thisObject );

time_t EDMRST_GetObjectModDate( serverHandle        svrHdl,
                                restorableObjectPtr thisObject );

u_hyper EDMRST_GetObjectSize( serverHandle        svrHdl,
                              restorableObjectPtr thisObject );

BackupStatus  EDMRST_GetObjectStatus( serverHandle        svrHdl,
                                      restorableObjectPtr thisObject
                                                                   );

char EDMRST_GetWorkItemType( serverHandle        svrHdl,
                             restorableObjectPtr thisObject );

const char *EDMRST_GetWorkItemBIC( serverHandle        svrHdl,
                                   restorableObjectPtr thisObject );

/*********************************************************************
 * Is Object Markable
 *
 *  This routine determines if the current user is able to restore the
 *  given object.
 *
 * Parameters:
 *
 *  svrHdl     (I) - A pointer to this user's client handle for the
 *                   Restore Engine (server) connection.
 *  thisObject (I) - The restoral object
 *  markable   (O) - TRUE/FALSE indicating markable or not
 *
 *********************************************************************/
```

```
eerrno_ty EDMRST_IsObjectMarkable( serverHandle          svrHdl,
                                   const restorableObjectPtr
                                                         thisObject,
                                   boolean_ty           *markable );

/*****************************************************************
 * Is Object Searchable
 *
 *  This routine determines if a top level object supports the Find
                                                          function.
 *
 * Parameters:
 *
 *  svrHdl      (I) - A pointer to this user's client handle for the
 *                    Restore Engine (server) connection.
 *  thisObject (I) - The restoral object to query
 *  searchable (O) - TRUE/FALSE indicating searchable or not
 *
 * Return Codes:
 *       E_SUCCESS                - operation completed successfully
 *        EP_RB_RECOVER_BAD_ARGS
 *        EP_RB_RECOVER_RPC_FAIL
 *        EP_RB_RECOVER_SERVER_FAIL
 *        EP_RB_RECOVER_INVALOP          -If another request active
 *
 *****************************************************************
                                                               */
eerrno_ty EDMRST_IsObjectSearchable( serverHandle        svrHdl,
                                     const restorableObjectPtr
                                                         thisObject,
                                     boolean_ty
                                                       *searchable );
/*****************************************************************/


/*****************************************************************
 * Get Symmetrix Support
 *
 *  This routine determines if a top level object supports restoral
                                                      thru a Symm
 *
 * Parameters:
 *
 *  svrHdl      (I) - A pointer to this user's client handle for the
 *                    Restore Engine (server) connection.
 *  thisObject (I) - The restoral object to query
 *  symm_restorable (O) - TRUE/FALSE indicating restorable over Symm (
                                                            SCSI)
 *
 * Return Codes:
 *       E_SUCCESS                - operation completed successfully
 *        EP_RB_RECOVER_BAD_ARGS
 *        EP_RB_RECOVER_RPC_FAIL
 *        EP_RB_RECOVER_SERVER_FAIL
 *        EP_RB_RECOVER_INVALOP          -If another request active
 *
 *****************************************************************
                                                               */
eerrno_ty EDMRST_GetSymmRestoreOption( serverHandle      svrHdl,
                                       const restorableObjectPtr
                                                         thisObject,
                                       boolean_ty
                                                    *symm_restorable );
```



```
/*****************************************************************
 * Get Network Support
 *
 *  This routine determines if a top level object supports restoral
                                                            thru
 *  the network
 *
 * Parameters:
 *
 *  svrHdl      (I) - A pointer to this user's client handle for the
 *                    Restore Engine (server) connection.
 *  thisObject (I) - The restoral object to query
 *  net_restorable (O) - TRUE/FALSE indicating restorable over Symm (
                                                            SCSI)
 *
 * Return Codes:
 *       E_SUCCESS                - operation completed successfully
 *        EP_RB_RECOVER_BAD_ARGS
 *        EP_RB_RECOVER_RPC_FAIL
 *        EP_RB_RECOVER_SERVER_FAIL
 *        EP_RB_RECOVER_INVALOP          -If another request active
 *
 *****************************************************************
                                                               */
eerrno_ty EDMRST_GetNetworkRestoreOption( serverHandle   svrHdl,
                                          const restorableObjectPtr
                                                         thisObject,
                                          boolean_ty
                                                    *net_restorable );
/*****************************************************************/

/*****************************************************************
 * Find Routines:
 *
 *  These routines allow the user to find restorable objects.
                                                          Returned
 *  is an array of found objects and an array of backup times
                                                          associated
 *  with the objects.  These arrays are 1-to-1.  That is,
                                                        the nth Object
 *  was backed up at the nth time.
 *
 *  This operation is performed asynchronously by the Restore Engine,
                                                           and the
 *  first API function, EDMRST_FindRestorableObjects,
                                                       is used to start the
 *  'find'.
     EDMRST_GetFindResults is used to teet for completion of the find,
 *  and receive the results (parts of, at least) if it is done.
 *
 * EDMRST_FindRestorableObjects Parameters:
 *
 *  svrHdl          (
                  I) - A pointer to this user's client handle for the
 *                       Restore Engine (server) connection.
 *  searchCriteria (I) - The criteria used for the search
 *
 *****************************************************************/
eerrno_ty EDMRST_FindRestorableObjects(
                                       serverHandle          svrHdl,
                                       EBREC_SearchCriteriaRec
                                                      *searchCriteria);
```

```
/*****************************************************************
 * EDMRST_GetFindResults Parameters:
 *
 *   svrHdl          (
 *                I)  - a pointer to this user's client handle for the
 *                       Restore Engine (server) connection.
 *   interrupt       (I)  - requests cancellation of the find (if TRUE)
 *   maxEntries      (
 *                I)  - the maximum number of found objects to return
 *   objects         (
 *                O)  - a pre-allocated array to return the objects in
 *   times           (
 *             O)  - a pre-allocated array to return the backup times in
 *   numberEntries (
 *                O)  - the real number of objects returned in the array
 *   cookie          (IO) - a place holder for the list position
 *
 *****************************************************************/
eerrno_ty EDMRST_GetFindResults( serverHandle          svrHdl,
                                 boolean_ty            interrupt,
                                 const long            maxEntries,
                                 restorableObjectPtr *foundObjects,
                                 time_t               *times,
                                 long                 *numberEntries,
                                 long                 *cookie );

/*****************************************************************
 * MarkObject() and GetMarkResults()
 *
 *   MarkObject is passed a restorableObject and marks files for
 *   recovery based on the input criteria.
 *                                  It starts an asynchronous operation
 *   in the Restore Engine to perform the marking, and returns.
 *
 *   GetMarkResults is called to test for completion of the mark
 *                                                        operation,
 *   and receive results when it is done.
 *                                  It can also be used to interrupt
 *   the mark operation.
 *
 * MarkObject Parameters:
 *
 *   svrHdl          (
 *                I) - A pointer to this user's client handle for the
 *                     Restore Engine (server) connection.
 *   thisObject      (I) - The restoral object;
 *                                  can be a leaf object (e.g. a
 *                     file), or a container object (
 *                                                  e.g., a directory).
 *   time            (I) - (
 *                     optional) the backup time to perform the mark on --
 *                     if not specified,
 *                                  uses currrently selected backup; if
 *                     specified,
 *                                  leaves selected backup time unchanged
 *   allowBadfiles (I) - allows marking of files of state BADDATA.
 *   descend         (
 *             I) - Should mark operation descend to operate on the content
 *                  of container objects.
 *
 *****************************************************************/

eerrno_ty EDMRST_MarkObject( serverHandle          svrHdl,
                             restorableObjectPtr thisObject,
                             time_t               time,
                             boolean_ty           allowBadFiles,
```



```
                             boolean_ty           descend );

/*****************************************************************
 *
 * GetMarkResults Parameters:
 *
 *   svrHdl          (
 *                I) - A pointer to this user's client handle for the
 *                     Restore Engine (server) connection.
 *   interrupt       (I) - requests cancellation of the mark (if TRUE)
 *                     WARNING: If the operation is aborted,
 *                                                  the mark will be
 *                     left in an unknown state.  That is,
 *                                                  any one of the
 *                     descendants of the marked object may be
 *                                                  marked or not.
 *                     It is up to the caller to determine how to
 *                                                  proceed
 *                     afterwards.
 *   BadFilesCount (O) - returns the file count with BADDATA.
 *   PermDenyFilesCount (
 *                O) -- returns the file count with permission denied.
 *   fileMarked      (
 *             O) - return the total files marked after this mark occurred.
 *   dirMarked       (
 *             O) - return the total directories marked after this mark
 *                  occurred.
 *   otherMarked     (
 *             O) - return the total "other" files marked after this mark.
 *
 *****************************************************************
                                                             **/

eerrno_ty EDMRST_GetMarkResults( serverHandle         svrHdl,
                                 boolean_ty           interrupt,
                                 u_long               *BadFilesCount,
                                 u_long
                                                      *PermDenyFilesCount,
                                 u_long               *fileMarked,
                                 u_long               *dirMarked,
                                 u_long               *otherMarked );

/*****************************************************************
 * UnmarkObject() and GetUnmarkResults()
 *
 *   UnmarkObject operates like MarkObject,
 *                                  in that it is supported through
 *   two API calls -- UnmarkObject and GetUnmarkresults.
 *                                  Unmark starts an
 *   asynchronous operation in the Restore Engine to perform the
 *                                                        unmarking,
 *   and returns.
 *
 *   GetUnmarkResults is called to test for completion of the unmark
 *                                                        operation,
 *   and receive results when it is done.
 *                                  It can also be used to interrupt
 *   the unmark operation.
 *
 * UnmarkObject Parameters:
 *
 *   svrHdl          (
 *                I) - A pointer to this user's client handle for the
 *                     Restore Engine (server) connection.
 *   thisObject      (I) - The restoral object;
                                  can be a leaf object (e.g. a
```

```
1127    *                          file), or a container object (
                                                        e.g., a directory).
1128    *  time           (I) - (
                        optional) the backup time to perform the unmark on --
1129    *                        if not specified,
                                        uses currrently selected backup; if
1130    *                        specified,
                                        leaves selected backup time unchanged
1131    *  BadFilesOnly   (
                        I) - allows unmarking ONLY of files of state BADDATA.
1132    *  descend        (
                        I) - Should unmark operation descend to operate on the
1133    *                        content of container objects.
1134    ************************************************************************
                                                                        **/

1136    eerrno_ty EDMRST_UnmarkObject( serverHandle        svrHdl,
1137                                   restorableObjectPtr thisObject,
1138                                   time_t              time,
1139                                   boolean_ty          BadFilesOnly,
1140                                   boolean_ty          descend );

1142    /************************************************************************
1143    *
1144    * GetUnmarkResults Parameters:
1145    *
1146    *  svrHdl         (
                        I) - A pointer to this user's client handle for the
1147    *                        Restore Engine (server) connection.
1148    *  interrupt      (I) - requests cancellation of the unmark (if TRUE)
1149    *                        WARNING: If the operation is aborted,
                                                        the unmark will
1150    *                        be left in an unknown state.  That is,
                                                        any one of the
1151    *                        descendants of the unmarked object may be
                                                        marked or
1152    *                        not.
                                        It is up to the caller to determine how to
1153    *                        proceed afterwards.
1154    *  BadFilesCount (O) - returns the file count with BADDATA.
1155    *  fileMarked     (
                        O) - return the total files marked after this mark occurred.
1156    *  dirMarked      (
                        O) - return the total directories marked after this mark
1157    *                        occurred.
1158    *  otherMarked    (
                        O) - return the total "other" files marked after this mark.
1159    *
1160    ************************************************************************/

1162    eerrno_ty EDMRST_GetUnmarkResults( serverHandle svrHdl,
1163                                       boolean_ty   interrupt,
1164                                       u_long       *BadFilesCount,
1165                                       u_long       *fileMarked,
1166                                       u_long       *dirMarked,
1167                                       u_long       *otherMarked );

1169    /************************************************************************
1170    * IsObjectMarked:
1171    *
1172    *  This function determines if each of a set of restorable objects
                                                                has been
1173    *  marked for restoration.
                                It is intended to allow the user to determine the
1174    *  current restore markings for the restorable objects at the same
                                                                object tree
```



```
1175    *  hierarchy level,
                        i.e. objects that have the same parent restorableObject.
1176    *
1177    * Parameters:
1178    *
1179    *  svrHdl         (I) - A pointer to this user's client handle for the
1180    *                        Restore Engine (server) connection.
1181    *  numEntries     (I) - number of objects in bufPtrArray to be checked
1182    *  bufPtrArray    (I) - restorable objects to be checked for marks
1183    *  numChecked     (O) - number of objects in bufPtrArray found marked
1184    *  marked         (O) - array of booleans to receive individual marked(
                                                                1) /
1185    *                        unmarked(
                                        0) result for corresponding bufPtrArray entry
1186    ************************************************************************/

1188    eerrno_ty EDMRST_IsObjectMarked( serverHandle         svrHdl,
1189                                     const unsigned long numEntries,
1190                                     restorableObjectPtr *bufPtrArray,
1191                                     unsigned long       *numChecked,
1192                                     boolean_ty          *marked );

1194    /************************************************************************/

1196    eerrno_ty EDMRST_GetMarkedTotalSize( serverHandle svrHdl,
1197                                         u_hyper *totalSize );

1199    /************************************************************************
1200    * Get Necessary Media:
1201    *
1202    *  This function is provided to allow retrieval of the
1203    *  media necessary to restore the currently marked objects.
1204    *
1205    *  The cookie must be initialize to INIT_COOKIE on the first call to
                                                                this
1206    *  routine.
                        This routine will update the cookie to allow retrieval of more
1207    *  objects if there is more than *maxEntries*.  The cookie will be
1208    *  returned as DONE_COOKIE when there are no more to retrieve.
1209    *
1210    * Parameters:
1211    *  svrHdl         (
                        I) - a pointer to this user's client handle for the
1212    *                        Restore Engine (server) connection.
1213    *  maxEntries     (I) - the maximum number of media objects to return
1214    *  objects        (O) - a pre-allocated array to return the objects in
1215    *  numberEntries  (
                        O) - the real number of media objects returned in the array
1216    *  all            (I) - get all media, or necessary
1217    *  cookie         (IO) - a place holder for the list position
1218    *
1219    ************************************************************************/
1220    eerrno_ty EDMRST_GetNecessaryMedia( serverHandle   svrHdl,
1221                                        const short    maxEntries,
1222                                        mediaObjectPtr *objects,
1223                                        short          *numberEntries,
1224                                        boolean_ty     all,
1225                                        long           *cookie );

1227    /************************************************************************
1228    * Media Object Access Routines:
1229    *
1230    *  These routines retrieve information pertinent to a given Media
                                                                object.
1231    *
1232    * Parameters:
```

```
1233    *
1234    *  svrHdl      (I) - (
                    ignored) A pointer to this user's client handle for the
1235    *                    Restore Engine (server) connection.
1236    *  thisObject (I) - The media object
1237    *
1238    ***********************************************************************/
1239    const char *EDMRST_GetMediaVolid( serverHandle   svrHdl,
1240                                      mediaObjectPtr thisObject );

1242    long EDMRST_GetMediaSequenceNumber( serverHandle   svrHdl,
1243                                        mediaObjectPtr thisObject );

1245    const char *EDMRST_GetMediaBarcodeString( serverHandle   svrHdl,
1246                                        mediaObjectPtr  thisObject );

1248    const char *EDMRST_GetMediaTypeDescrip( serverHandle   svrHdl,
1249                                        mediaObjectPtr thisObject );

1251    const char *EDMRST_GetMediaTrail( serverHandle   svrHdl,
1252                                      mediaObjectPtr thisObject );

1254    const char *EDMRST_GetMediaTypeToken( serverHandle   svrHdl,
1255                                      mediaObjectPtr thisObject );

1257    const char *EDMRST_GetMediaComments( serverHandle   svrHdl,
1258                                      mediaObjectPtr thisObject );

1260    const char *EDMRST_GetMediaLocation( serverHandle   svrHdl,
1261                                      mediaObjectPtr thisObject );

1263    MediaStatus  EDMRST_GetMediaStatus( serverHandle   svrHdl,
1264                                        mediaObjectPtr thisObject );

1266    uchar_t EDMRST_GetMediaSide( serverHandle   svrHdl,
1267                                 mediaObjectPtr thisObject );




1271    /*****************************************************
1272     * Duplicate Media Access Routines
1273     * Inputs: Svr Handle - see above
1274     *         dup_number: the number of the duplicate wanted
1275     *         thiObject:  The media object to get the dups for...
1276     *****************************************************
1277     */
1278    short EDMRST_GetNumberOfDuplicates( serverHandle   svrHdl,
1279                                        mediaObjectPtr thisObject );


1282    const char *
1283    EDMRST_GetDuplicateVolid( serverHandle   svrHdl,
1284                              int dup_number,
1285                              mediaObjectPtr thisObject );

1287    long
1288    EDMRST_GetDuplicateSequenceNumber( serverHandle   svrHdl,
1289                                       int dup_number,
1290                                       mediaObjectPtr thisObject );

1292    const char *
1293    EDMRST_GetDuplicateBarcodeString( serverHandle   svrHdl,
1294                                      int dup_number,
1295                                      mediaObjectPtr thisObject );

1297    const char *
```



```
1298    EDMRST_GetDuplicateTypeDescrip( serverHandle   svrHdl,
1299                                    int dup_number,
1300                                    mediaObjectPtr thisObject );


1303    const char *
1304    EDMRST_GetDuplicateTypeToken( serverHandle   svrHdl,
1305                                  int dup_number,
1306                                  mediaObjectPtr thisObject );


1309    MediaStatus
1310    EDMRST_GetDuplicateStatus( serverHandle   svrHdl,
1311                               int dup_number,
1312                               mediaObjectPtr thisObject );
1313    const char *
1314    EDMRST_GetDuplicateLocation( serverHandle   svrHdl,
1315                                 int dup_number,
1316                                 mediaObjectPtr thisObject );


1318    /*****************************************************************
1319     * Restoral Management Functions:
1320     *
1321     *  These functions are provided to allow:
1322     *       - creation of submit objects,
                                          which define the set of objects to be
1323     *         restored and the scripts to be run before and after
                                                              restoration,
1324     *       - starting the restoral of a submit object.
1325     *       - polling of the status of an ongoing restore,
                                                    including the ability to
1326     *         interrupt the restore,
                                                  and to receive information necessary to
1327     *         query the user for input needed for the pre-restore or
                                                              post-restore
1328     *         scripts, suspending, restarting,
1329     *       - supply of user responses to pre- and post- restore script
                                                              queries
1330     *
1331     * The following functions comprise restoral management:
1332     *
1333     *       EDMRST_Submit
1334     *       EDMRST_GetSubmitResults
1335     *        EDMRST_Start
1336     *        EDMRST_GetRestoreFeedback
1337     *        EDMRST_GetQuestion
1338     *        EDMRST_SetUserAnswer
1339     *
1340     *****************************************************************
1341     * Submit
1342     *
1343     * This function starts the creation or update of a submit object from
                                                                  the
1344     * currently marked restorable objects.
                                                  Its completion is tested for with
1345     * EDMRST_GetSubmitResults.
                                                  The returned submit object ID is passed to
1346     * EDMRST_Start to begin execution of the restore.
1347     *
1348     * Parameters:
1349     *
1350     * svrHdl        (I) - A pointer to this user's client handle for
1351     *                     the Restore Engine (server) connection.
1352     * policy        (I) - The overwrite policy to use
1353     * inPlace       (
                        I) - Flag if the restoral is to be in original locations
```

```
1354    * hostName      (I) - host to restore to (only if inPlace == False)
1355    * directory     (I) - directory to restore to (
                                                only if inPlace == False)
1356    * transport     (
                I) - Indicator of transport the restoral is to be over (SCSI
1357    *                    or network)
1358    * submitObjID    (
                I) - ID of an existing submit object which is to be added to
1359    *
1360    * Return Codes:
1361    *       E_SUCCESS                   - operation completed successfully
1362    *       EP_RB_RECOVER_BAD_ARGS
1363    *       EP_RB_RECOVER_NOMEM
1364    *       EP_RB_RECOVER_RPC_FAIL
1365    *       EP_RB_RECOVER_SERVER_FAIL
1366    *       EP_RB_RECOVER_INVALOP               -If another request still
                                                              executing
1367     ****************************************************************/
1368  eerrno_ty EDMRST_Submit( serverHandle            svrHdl,
1369                           const char              *hostName,
1370                           const OverwritePolicy   policy,
1371                           const boolean_ty        inPlace,
1372                           const char              *directory,
1373                          const RestoreTransport transport,
1374                          unsigned int            submitObjID,
1375                           EDMRST_submit_args      *submitArgs);
1376
1377  /****************************************************************
1378   * GetSubmitResults
1379   *
1380   * This function tests for completion of an EDMRST_Submit call,
                                                          with the
1381   * option of cancelling the submit.
1382   *
1383   * Parameters:
1384   *
1385   * svrHdl        (I) - A pointer to this user's client handle for
1386   *                     the Restore Engine (server) connection.
1387   * interrupt     (I) - Flag if the submit is to be canceled
1388   * submitObjID   (
                O) - ID of the submit object which describes the restore
1389   * objectsDone   (
                O) - number of objects -- total number in the submit object
1390   *                     if operation is complete,
                                or number processed so far if
1391   *                     submit operation is still executing (
                                                    INCOMPLETE status)
1392   *
1393   * Return Codes:
1394   *       E_SUCCESS                   - operation completed successfully
1395   *       EP_RB_RECOVER_RPC_INCOMPLETE    -If submit not done yet
1396   *       EP_RB_RECOVER_ABORT             -If submit successfully
                                                        interrupted
1397   *       EP_RB_RECOVER_FATALERR          -If submit failed.
                                                    Internal restore error
1398   *       EP_RB_RECOVER_BAD_ARGS
1399   *       EP_RB_RECOVER_NOMEM
1400   *       EP_RB_RECOVER_RPC_FAIL
1401   *       EP_RB_RECOVER_SERVER_FAIL
1402   *       EP_RB_RECOVER_INVALOP           -If another request still
                                                              executing
1403    ****************************************************************/
1404  eerrno_ty EDMRST_GetSubmitResults( serverHandle     svrHdl,
1405                                     const boolean_ty interrupt,
1406                                     unsigned int     *submitObjID,
1407                                     unsigned long    *objectsDone );
```



```
1410  /****************************************************************
1411   * Start
1412   *
1413   * This function begins execution of the restoral of the objects in a
1414   * submit object.  Its progress and requests for operator input are
1415   * received via EDMRST_GetRestoreFeedback.
1416   *
1417   * Parameters:
1418   *
1419   * svrHdl        (I) - A pointer to this user's client handle for
1420   *                           the Restore Engine (server) connection.
1421   * submitObjID   (
                  I) - ID of the submit object that describes the restore
1422   *
1423   * Return Codes:
1424   *       E_SUCCESS                   - operation completed successfully
1425   *       EP_RB_RECOVER_BAD_ARGS
1426   *       EP_RB_RECOVER_NOMEM
1427   *       EP_RB_RECOVER_RPC_FAIL
1428   *       EP_RB_RECOVER_SERVER_FAIL
1429   *       EP_RB_RECOVER_INVALOP               -If another request still
                                                              executing
1430    ****************************************************************/
1431  eerrno_ty EDMRST_Start( serverHandle  svrHdl,
1432                          unsigned int submitObjID );
1433
1434  /****************************************************************
1435   * GetRestoreFeedback
1436   *
1437   * This function is used to poll for the status of an ongoing restore,
                                                                and
1438   * includes the ability to interrupt the restore,
                                        and to receive information
1439   * necessary to query the user for input needed for the pre-restore or
1440   * post-restore scripts.
1441   *
1442   * Parameters:
1443   *
1444   * svrHdl        (I) - A pointer to this user's client handle for
1445   *                           the Restore Engine (server) connection.
1446   * quitRestore   (I) - Flag if the restoral is to be stopped
1447   * currentState  (
                  O) - Pointer to storage to receive the state of the restore
1448   * feedbackPtr   (
                  IO) - Pointer to structure to receive restore feedback data
1449   *
1450   * Return Codes:
1451   *       E_SUCCESS                   - If restore operation completed
                                                          successfully
1452   *       EP_RB_RECOVER_RPC_INCOMPLETE    -If restore not done yet
1453   *       EP_RB_RECOVER_ABORT             -If restore successfully
                                                        interrupted
1454   *       EP_RB_RECOVER_EXECUTEFAILED     -If restore execution failed
1455   *       EP_RB_RECOVER_PREFAILED                 -If pre-restore
                                                  script execution failed
1456   *       EP_RB_RECOVER_POSTFAILED        -If post-restore script
                                                      execution failed
1457   *       EP_RB_RECOVER_CLIENT_FAIL       -If problem with restore
                                                        target host
1458   *       EP_RB_RECOVER_BAD_ARGS
1459   *       EP_RB_RECOVER_NOMEM
1460   *       EP_RB_RECOVER_RPC_FAIL
1461   *       EP_RB_RECOVER_SERVER_FAIL
1462   *       EP_RB_RECOVER_INVALOP           -If another request still
```

```
                                                                executing
1463    ***************************************************************/

1465    eerrno_ty EDMRST_GetRestoreFeedback( serverHandle     svrHdl,
1466                                         const boolean_ty quitRestore,
1467                                         RERunningState   *currentState,
1468                                         feedbackObjectPtr feedbackPtr );

1470    /***************************************************************
1471     * GetQuestion
1472     *
1473     * This function is used to fetch the data needed to query the user
                                                                  during a
1474     * pre-restore or post-restore script execution.
1475     *
1476     * Parameters:
1477     *
1478     * svrHdl    (I) - A pointer to this user's client handle for
1479     *                 the Restore Engine (server) connection.
1480     * queryPtr (O) - Pointer to the object containing the question data.
1481     *
1482     * Return Codes:
1483     *       E_SUCCESS                 - operation completed successfully
1484     *       EP_RB_RECOVER_BAD_ARGS
1485     *       EP_RB_RECOVER_RPC_FAIL
1486     *       EP_RB_RECOVER_SERVER_FAIL
1487     *       EP_RB_RECOVER_INVALOP           -If no question awaiting
                                                                    answer
1488    ***************************************************************/

1490    eerrno_ty EDMRST_GetQuestion( serverHandle      svrHdl,
1491                                  queryObjectPtr    queryPtr );

1493    /***************************************************************
1494     * SetUserAnswer
1495     *
1496     * This function is used to return user input requested via the
                                                              feedbackPtr
1497     * parameter output of the GetRestoreFeedback function call.
1498     *
1499     * Parameters:
1500     *
1501     * svrHdl    (I) - A pointer to this user's client handle for
1502     *                 the Restore Engine (server) connection.
1503     * QueryPtr  (
1504     *            I) - Pointer to structure containing the query and response
                          data.
1505     * answer    (I) - pointer to text string response to question.
1506     * more      (
1507     *            I) - indicator that there will be more answers to this question
1508     *
1509     * Return Codes:
1510     *       E_SUCCESS                 - operation completed successfully
1511     *       EP_RB_RECOVER_BAD_ARGS
1512     *       EP_RB_RECOVER_NOMEM
1513     *       EP_RB_RECOVER_RPC_FAIL
1514     *       EP_RB_RECOVER_SERVER_FAIL
1515     *       EP_RB_RECOVER_INVALOP           -If no question awaiting
                                                                    answer
1516     *
1517     ***************************************************************/

1518    eerrno_ty EDMRST_SetUserAnswer( serverHandle     svrHdl,
1519                                    queryObjectPtr  queryPtr,
1520                                    const char      *answer,
1521                                    boolean_ty      more );
```



```
1524    /***************************************************************
1525     * Query Object Access Routines:
1526     *
1527     *  These routines retrieve portions of a specified Query object.
                                                                  A query
1528     *  object is returned by EDMRST_GetQuestion,
                                                    and is used to get a response
1529     *  from the Restore API 'operator' after a restore has been started.
1530     *
1531     * Parameters common to all these functions:
1532     *  svrHdl      (I) - A pointer to this user's client handle for the
1533     *                    Restore Engine (
                                                          server) connection. Must be valid.
1534     *  thisObject (I) - The query object
1535     *
1536     * parameters specific to only one function:
1537     *  minlen      (O) - minumum response length
                                                                  0 on errs
1538     *  maxlen      (O) - maximum response length
1539     *  isset       (
                          O) - boolean_ty indicating this is the default response
1540     *  cookie     (IO) - placeholder in list of possible choices
1541     *
1542     * Returns one of the following:
1543     *  int                 question type or number of possible responses;
                                                                  0 on errs
1544     *  const char *        ptr to a text string in the query object,
                                                                  NULL on errs
1545     *  eerrno_ty           E_SUCCESS; or EP_RB_RECOVER_xxx on errors
1546     *
1547     ***************************************************************/

1549    int EDMRST_GetQuestionType (serverHandle   svrHdl,
1550                                queryObjectPtr thisObject );

1552    eerrno_ty EDMRST_GetQuestionAnswerSize (serverHandle    svrHdl,
1553                                            queryObjectPtr  thisObject,
1554                                            int             *minlen,
1555                                            int             *maxlen );

1557    int EDMRST_GetQuestionNumChoices( serverHandle   svrHdl,
1558                                      queryObjectPtr thisObject );

1560    const char * EDMRST_GetQuestionNextChoice(
                                              serverHandle          svrHdl,
1561                                            queryObjectPtr
                                                          thisObject,
1562                                            int                 *isset,
1563                                            long
                                                              *cookie );

1565    const char * EDMRST_GetQuestionInvalidChars(
                                            serverHandle        svrHdl,
1566                                          queryObjectPtr
                                                        thisObject );

1568    const char * EDMRST_GetQuestionHeaderText(
                                           serverHandle         svrHdl,
1569                                         queryObjectPtr
                                                        thisObject );

1571    const char * EDMRST_GetQuestionText( serverHandle       svrHdl,
1572                                         queryObjectPtr     thisObject );
```

```
/*****************************************************************/
/* EDMRST_GetHostPlatformType                                    */
/*                                                               */
/*       This function returns the platform type of the          */
/*       specified host.                                         */
/*                                                               */
/* Parameters:                                                   */
/*       svrHdl          -  (I) A pointer to this user's client  */
/*                          handle for the Restore Engine        */
/*                          connection.                          */
/*       pHostName       - (I) ptr to hostname string;           */
/*       pType           - (O) ptr to host platform type, which  */
/*                          can be one of the following valid    */
/*                          types:                               */
/*                                                               */
/*                             BUCFG_PLATFORM_UNIX               */
/*                             BUCFG_PLATFORM_OS2                */
/*                             BUCFG_PLATFORM_WNT                */
/*                             BUCFG_PLATFORM_NETWARE            */
/*                                                               */
/* Return Codes:                                                 */
/*       E_SUCCESS:                                              */
/*                          - operation successful, *pType will  */
/*                          contain one of the valid types       */
/*                          described above.                     */
/*                                                               */
/*       EP_RB_RECOVER_BAD_ARGS                                  */
/*       EP_RB_RECOVER_RPC_FAIL                                  */
/*       EP_RB_RECOVER_SERVER_FAIL                               */
/*       EP_RB_RECOVER_INVALOP   - If another request active     */
/*                                                               */
/*****************************************************************/
eerrno_ty
EDMRST_GetHostPlatformType( serverHandle      svrHdl,
                            const char        *pHostName,
                            BuCfgPlatformType *pType );


/*****************************************************************
 * EDMRST_GetBackupTimesSupport
 *
 * Function Description:
 *      Determine if the specified top level object can be restored
 *       from multiple backup times.
 *
 * Parameters:
 *       svrHdl       (I) - A pointer to this user's client handle for
 *                          the Restore Engine (server) connection.
 *       thisObject  (I) - The restoral object
 *       isSupported (
 *                    O) - TRUE/FALSE that restores from multiple backup planes
 *                         is/is not supported
 *
 * Return Codes:
 *       E_SUCCESS                  - operation completed successfully
 *       EP_RB_RECOVER_BAD_ARGS
 *       EP_RB_RECOVER_RPC_FAIL
 *       EP_RB_RECOVER_SERVER_FAIL
 *       EP_RB_RECOVER_INVALOP           -If another request active
 *
 *****************************************************************
                                                                */

eerrno_ty EDMRST_GetBackupTimesSupport(  serverHandle     svrHdl,
                                         restorableObjectPtr
```



```
                                                         thisObject,
                                         boolean_ty      *isSupported
                                                                  );

/*****************************************************************/
/* Query available backups                                       */
/*                                                               */
/*       There are 4 API functions in this group:                */
/*                EDMRST_IsTherePrevBackup()                     */
/*                EDMRST_IsThereNextBackup()                     */
/*                EDMRST_IsTherePrevBackupForTime()              */
/*                EDMRST_IsThereNextBackupForTime()              */
/*                                                               */
/*       These functions allow the caller to query whether or    */
/*       not a previous or next backup exists relative to either */
/*       the currently selected backup, or the backup of the     */
/*       specified time. Additionally, the caller can specify    */
/*       selction flags such as complete backups only.           */
/*****************************************************************/

/*****************************************************************
 * EDMRST_IsTherePrevBackup
 *
 * Function Description:
 *      Determine if a backup exists prior to the backup that is
 *       currently selected.
 *
 * Parameters:
 *       svrHdl   (I) - A pointer to this user's client handle for
 *                      the Restore Engine (server) connection.
 *       flags    (
 *                   I) - Backup constraint flags: e.g., full-only/partial-ok
 *       isThere  (O) - TRUE/FALSE that requested backup does exist
 *
 * Return Codes:
 *       E_SUCCESS                  - operation completed successfully
 *       EP_RB_RECOVER_BAD_ARGS
 *       EP_RB_RECOVER_RPC_FAIL
 *       EP_RB_RECOVER_SERVER_FAIL
 *       EP_RB_RECOVER_INVALOP           -If another request active
 *       EP_RB_RECOVER_NO_SAVESET        - when errors occur accessing
                                                                savesets
 *       EP_RB_RECOVER_NO_CATALOG        - when errors occur accessing
                                                                catalogs
 *
 *****************************************************************
                                                                */

eerrno_ty
EDMRST_IsTherePrevBackup( serverHandle    svrHdl,
                          u_long                  flags,
                          boolean_ty      *isThere );

/*****************************************************************
 * EDMRST_IsThereNextBackup
 *
 * Function Description:
 *      Determine if a backup exists after the backup that is
 *       currently selected.
 *
 * Parameters:
 *       svrHdl   (I) - A pointer to this user's client handle for
 *                      the Restore Engine (server) connection.
 *       flags    (
 *                   I) - Backup constraint flags: e.g., full-only/partial-ok
 *       isThere  (O) - TRUE/FALSE that requested backup does exist
```

```
1697    *
1698    * Return Codes:
1699    *       E_SUCCESS                   - operation completed successfully
1700    *        EP_RB_RECOVER_BAD_ARGS
1701    *        EP_RB_RECOVER_RPC_FAIL
1702    *        EP_RB_RECOVER_SERVER_FAIL
1703    *        EP_RB_RECOVER_INVALOP              -If another request active
1704    *       EP_RB_RECOVER_NO_SAVESET            - when errors occur accessing
                                                          savesets
1705    *       EP_RB_RECOVER_NO_CATALOG            - when errors occur accessing
                                                          catalogs
1706    *
1707    ************************************************************************
                                                                          */


1709    eerrno_ty
1710    EDMRST_IsThereNextBackup( serverHandle   svrHdl,
1711                              u_long         flags,
1712                              boolean_ty     *isThere );


1714    /*************************************************************
1715     * EDMRST_IsTherePrevBackupForTime
1716     *
1717     * Function Description:
1718     *       Determine if a backup exists prior to the specified time
1719     *
1720     * Parameters:
1721     *       svrHdl  (I) - A pointer to this user's client handle for
1722     *                     the Restore Engine (server) connection.
1723     *       thisTime(I) - Time for the query
1724     *       flags   (
1725     *                 I) - Backup constraint flags: e.g., full-only/partial-ok
1725     *       isThere (O) - TRUE/FALSE that requested backup does exist
1726     *
1727     * Return Codes:
1728     *       E_SUCCESS                   - operation completed successfully
1729     *        EP_RB_RECOVER_BAD_ARGS
1730     *        EP_RB_RECOVER_RPC_FAIL
1731     *        EP_RB_RECOVER_SERVER_FAIL
1732     *        EP_RB_RECOVER_INVALOP              -If another request active
1733     *       EP_RB_RECOVER_NO_SAVESET            - when errors occur accessing
                                                          savesets
1734    *       EP_RB_RECOVER_NO_CATALOG            - when errors occur accessing
                                                          catalogs
1735    *
1736    ************************************************************************
                                                                          */


1738    eerrno_ty
1739    EDMRST_IsTherePrevBackupForTime( serverHandle           svrHdl,
1740                                     const time_t           thisTime,
1741                                     u_long                 flags,
1742                                     boolean_ty             *isThere );


1744    /*************************************************************
1745     * EDMRST_IsThereNextBackupForTime
1746     *
1747     * Function Description:
1748     *       Determine if a backup exists after to the specified time
1749     *
1750     * Parameters:
1751     *       context (I) - Pointer to the restore context
1752     *       thisTime(I) - Time for the query
1753     *       flags   (
1753     *                 I) - Backup constraint flags: e.g., full-only/partial-ok
1754     *       isThere (O) - TRUE/FALSE that requested backup does exist
```



```
1755    *
1756    * Return Codes:
1757    *       E_SUCCESS                   - operation completed successfully
1758    *        EP_RB_RECOVER_BAD_ARGS
1759    *        EP_RB_RECOVER_RPC_FAIL
1760    *        EP_RB_RECOVER_SERVER_FAIL
1761    *        EP_RB_RECOVER_INVALOP              -If another request active
1762    *       EP_RB_RECOVER_NO_SAVESET            - when errors occur accessing
                                                          savesets
1763    *       EP_RB_RECOVER_NO_CATALOG            - when errors occur accessing
                                                          catalogs
1764    *
1765    ************************************************************************
                                                                          */


1767    eerrno_ty
1768    EDMRST_IsThereNextBackupForTime( serverHandle           svrHdl,
1769                                     const time_t           thisTime,
1770                                     u_long                 flags,
1771                                     boolean_ty             *isThere );


1773    /*************************************************************
1774     * Query WIProgress Access Routines:
1775     *
1776     *  These routines retrieve portions of a specified WIProgress object.
                                                                          A
1777     *  WIProgress object is returned by EDMRST_XXXXXXXX,
                                                  and is used to get
1778     *  the first WIProgress from a EDMStats objecti which is return from
                                                                          the
1779     *  EMMRST_XXXXXXXX call.
1780     *
1781     * Parameters common to all these functions:
1782     *  svrHdl      (I) - A pointer to this user's client handle for the
1783     *                    Restore Engine (
                                          server) connection. Must be valid.
1784     *  thisObject (I) - The query object
1785     *
1786     * Returns one of the following:
1787     *
1788     *************************************************************/


1790    const char * EDMRST_GetWIObjectName(
                                  serverHandle svrHdl, WIProgressPtr thisObject);
1791    const char * EDMRST_GetWIObjectTrailName(
                                  serverHandle svrHdl, WIProgressPtr thisObject);
1792    const char * EDMRST_GetWIObjectTrailSetName(
                                  serverHandle svrHdl, WIProgressPtr thisObject);
1793    const char * EDMRST_GetWIObjectTemplateName(
                                  serverHandle svrHdl, WIProgressPtr thisObject);
1794    const char * EDMRST_GetWIObjectClientName(
                                  serverHandle svrHdl, WIProgressPtr thisObject);
1795    const char * EDMRST_GetWIObjectServerName(
                                  serverHandle svrHdl, WIProgressPtr thisObject);
1796    const char * EDMRST_GetWIObjectMediaType(
                                  serverHandle  svrHdl, WIProgressPtr thisObject);
1797    const char * EDMRST_GetWIObjectUserId(
                                  serverHandle svrHdl, WIProgressPtr thisObject);
1798    const char * EDMRST_GetWIObjectLevel(
                                  serverHandle svrHdl, WIProgressPtr thisObject);
1799    const char * EDMRST_GetWIObjectType(
                                  serverHandle svrHdl, WIProgressPtr thisObject);
1800    u_long EDMRST_GetObjectTimeStarted(
                                  serverHandle svrHdl, WIProgressPtr thisObject);
1801    u_long EDMRST_GetObjectCurrTime(
                                  serverHandle svrHdl, WIProgressPtr thisObject);
```

```
1802  u_long EDMRST_GotObjectTotalKbytesSofar(
                              serverHandle svrHdl, WIProgressPtr thisObject);
1803  u_long EDMRST_GetObjectTotalFiles(
                              serverHandle svrHdl, WIProgressPtr thisObject);
1804  u_long EDMRST_GetObjectTotalBadFiles(
                              serverHandle svrHdl, WIProgressPtr thisObject);
1805  u_long EDMRST_GetObjectCurrKbytesSofar(
                              serverHandle svrHdl, WIProgressPtr thisObject);
1806  u_long EDMRST_GetObjectCurrTimeSlice(
                              serverHandle svrHdl, WIProgressPtr thisObject);
1807  u_long EDMRST_GetObjectCurrFiles(
                              serverHandle svrHdl, WIProgressPtr thisObject);
1808  u_long EDMRST_GetObjectTotalFilesExpected(
                              serverHandle svrHdl, WIProgressPtr thisObject);
1809  u_long EDMRST_GetObjectTotalKbExpected(
                              serverHandle svrHdl, WIProgressPtr thisObject);
1810  u_long EDMRST_GetObjectItemStatus(
                              serverHandle svrHdl, WIProgressPtr thisObject);
1811  int EDMRST_GetObjectOperationType(
                              serverHandle svrHdl, WIProgressPtr thisObject);
1812  int EDMRST_GetObjectCompleted(
                              serverHandle svrHdl, WIProgressPtr thisObject);
1813  int EDMRST_IsLastObject(serverHandle svrHdl, WIProgressPtr thisObject);
1814  WIProgressPtr EDMRST_GetNextObject(
                              serverHandle svrHdl, WIProgressPtr thisObject);
1815  WIProgressPtr EDMRST_GetFirstObject(
                         serverHandle svrHdl,feedbackObjectPtr thisObject);

1817  EDMProgressPtr
1818  EDMRST_GetFirstEDMObject(
                        serverHandle svrHdl, feedbackObjectPtr thisObject);
1819  EDMProgressPtr
1820  EDMRST_GetNextEDMObject(
                              serverHandle svrHdl, EDMProgressPtr thisObject);
1821  u_long
1822  EDMRST_GetObjectEDMTimeStarted(
                              serverHandle svrHdl, EDMProgressPtr thisObject);
1823  u_long
1824  EDMRST_GetObjectEDMCurrTime(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1825  u_long
1826  EDMRST_GetObjectEDMTotalKbytesSofar(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1827  u_long
1828  EDMRST_GetObjectEDMTotalFiles(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1829  u_long
1830  EDMRST_GetObjectEDMTotalBadFiles(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1831  u_long
1832  EDMRST_GetObjectEDMCurrTimeSlice(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1833  u_long
1834  EDMRST_GetObjectEDMCurrKbytesSofar(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1835  u_long
1836  EDMRST_GetObjectEDMCurrFiles(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1837  u_long
1838  EDMRST_GetObjectEDMActive(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1839  u_long
1840  EDMRST_GetObjectEDMTotal(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1841  u_long
1842  EDMRST_GetObjectEDMFailed(
```



```
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1843  u_long
1844  EDMRST_GetObjectEDMSuccessful(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1845  u_long
1846  EDMRST_GetObjectEDMTotalFilesExpected(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1847  u_long
1848  EDMRST_GetObjectEDMTotalKBExpected(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1849  int
1850  EDMRST_GetObjectEDMOperationType(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1851  int
1852  EDMRST_GetObjectEDMCompleted(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1853  u_long
1854  EDMRST_GetObjectEDMStatus(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1855  const char *
1856  EDMRST_GetObjectEDMHostName(
                              serverHandle svrHdl,EDMProgressPtr thisObject);
1857  /** Notify Access Routines */
1858  void*
1859  EDMRST_GetFirstNotifyObject(
                         serverHandle svrHdl,feedbackObjectPtr thisObject);
1860  void*
1861  EDMRST_GetNextNotifyObject(serverHandle svrHdl,void* thisObject);
1862  const char *
1863  EDMRST_GetNotifyMessageText(serverHandle  svrHdl,void* thisObject);
1864  int
1865  EDMRST_GetNotifyMessageType(serverHandle svrHdl,void* thisObject);
1866  int
1867  EDMRST_GetNotifySourceModule(serverHandle svrHdl,void* thisObject);
1868  int
1869  EDMRST_GetNotifyLevel(serverHandle svrHdl,void* thisObject);
1870  int
1871  EDMRST_GetNotifyMessageLength(serverHandle svrHdl,void* thisObject);
1872  int
1873  EDMRST_IsLastNotifyObject(serverHandle svrHdl,void* thisObject);


1876  /*****************************************************************
1877   * SetRecxDirectives:
1878   *
1879   *  This routine returns sends the Filename and path plus hostname
1880   *     of the recx directives file, which was created by the command
1881   *     eb_dc_restore, to the server which then processes the recx
1882   *     directives
1883   *
1884   * Parameters:
1885   *  svrHdl     (I) - A pointer to this user's client handle for the
1886   *                   Restore Engine (server) connection.
1887   *  template  (O) - The name of the local recx file
1888   *  alternate (O) - the name of this host so the file can be tranfered
1889   *
1890   *****************************************************************/
1891  eerrno_ty EDMRST_SetRecxDirectives( serverHandle     svrHdl,
1892                                      char             *filename,
1893                                      char             *hostname );


1896  /*****************************************************************
1897   * EDMRST_get_catalog_info:
1898   *
1899   *  This routine returns sends the fills the level string with the
```

```
1900    *  level for backup being restored
1901    *
1902    * Parameters:
1903    *  svrHdl              (
                            I) - A pointer to this user's client handle for the
1904    *                           Restore Engine (server) connection.
1905    *  backup_time      (I) - Time of the backup that is beeing looked at
1906    *  *level           (O) - The level of the backup for specified time
1907    *                         taken from catalog structure.  If not enough
1908    *                         memory has been allocated value will be "\0"
1909    *  *numrec             (
                            O) - The number of records for the specified backup
1910    *                         taken from catalog structure.   If not enough
1911    *                         memory has been allocated value will be "\0"
1912    *  *catType         (O) - The type of catalog for the specified bacup
1913    *                         taken from catalog structure.   If not enough
1914    *                         memory has been allocated value will be "\0"
1915    * Return Codes:
1916    *   EP_RB_RECOVER_BAD_ARGS - arguments passed in are null
1917    *   E_SUCCESS              - the fields have been filled in
1918    *                            and RPC succeeded
1919    *
1920    ************************************************************************/
1921   eerrno_ty
1922   EDMRST_getCatalogInfo(   serverHandle   svrHdl,
1923                            time_t          backup_time,
1924                            char            *level,
1925                            char            *numRec,
1926                            char            *catType);

1928   #endif   /* H_RESTOREAPI */
```

D2

```
1  /*****************************************************************
2   *       restoreRPC.h
3   *
4   * Defintions of data sent between client and server for Restore API.
5   *
6   * NOTE:
7   *
8   * This file is intended to be shared by both RPC independent code on
                                                                    the
9   * client and server,
                    and by RPC implementation specific code.  That is, it
10  * is intended to be inlcuded in RPC definition files -- .
                                                  x files for ONC RPC --
11  * and in normal 'C' code.
12  *
13  * This is so that the client and server code at all but the RPC call
                                                                  and RPC
14  * service levels are RPC mechanism independent.  HOWEVER,
                                                          some of these
15  * definitions are very similar to ONC RPC data types (
                                                  e.g. STRING and OPAQUE),
16  * and for another RPC implementation,
                                hopefully only a redefinition of those
17  * two macros would be needed.
18  *****************************************************************/

20  #ifndef H_RESTORE_RPC_DATA
21  #define H_RESTORE_RPC_DATA

23  /* Don't redefine this stuff if restore_engine.
                                                  h has already been included */
24  #ifndef _RESTORE_ENGINE_H_RPCGEN

26  /* need to use string <> for .x file, char * for .h */
27  /* This doesn't work for some reason:  #if RPC_HDR */
28  #ifdef IN_DOTX
29  #define STRING(x) string x<>
30  #define OPAQUE(x) opaque x<>
31  #else
32  #define STRING(x) char *x
33  #define OPAQUE(x) struct { u_int length; char *data; } x
34  #endmif
35  #define RAW_NETWORK 0
36  #define PLUGIN 1

38  /*
39   * Bit field values for the Top Level Object "flags" field
40   */

42  typedef u_int     RSTRPC_bool;

44  typedef unsigned long     RSTRPC_backup_flags_ty;

46  typedef long      RSTRPC_time_ty;

48  typedef int       RSTRPC_enum_ty;

50  struct RSTRPC_u_hyper {
51          unsigned long   high;
52          unsigned long   low;
53  };

55  /* generic linked list structure definition: */
56  /* Linked lists used by the Restore Service Library must follow this
                                                                  format --
57   * that is,
```



```
58   *       must begin with the link to the next list entry.  The struct_ptr
     * entry does NOT have to be a char *, but can be anything.
                                                        even a whole struct.
59   */
60  typedef char *    struct_ptr;

62  struct RSTRPC_list_ty {
63          struct RSTRPC_list_ty   *next;
64          struct_ptr              objPtr;
65  };

67  struct RSTRPC_time_list {
68          struct RSTRPC_time_list         *next;
69          RSTRPC_time_ty          time;
70  };

72  struct RSTRPC_name_list {
73          struct RSTRPC_name_list         *next;
74          STRING                  (name);
75  };


78  /* Structures for use in passing file name of the direct connect
79   * directives
80   */
81  struct RSTRPC_recx_file_info {

83          STRING (filename);
84          STRING (hostname);
85  };




90  /*
91   * Definition of restorableObject data components for shared use
                                                                between
92   * generic (
           ie not app-specific) client and server portions of Restore API.
93   *
94   * The restorableObject is presented to the users
95   * of the Restore API as an opaque object,
                                                  meaning that its content and
96   * format are not known and not relevant to anyone except the Restore
97   * API internals. However, for Restore API internals,
                                                      it must be shared by
98   * programs on the client and server side of the Restore API.
                                                          The following
99   * structure is designed to be shared by both sides of the API,
                                                              and to be
100  * backup-application independent.
                                                  That is why the application-specific data
101  * within this structure is treated as opaque (
                                                      long length and void *).
102  */

104 enum RSTRPC_BackupStatus {
105         RSTRPC_Backup_Good,
106         RSTRPC_Backup_Bad,
107         RSTRPC_Backup_Expired,
108         RSTRPC_Backup_Child_Without_Data
109 };

111 enum RSTRPC_ObjectLevel {
112         RSTRPC_tlo_type = 1,
113         RSTRPC_container_type,
```

```
1          RSTRPC_leaf_type
1        };


1        /* common part of all restorable objects: */

1        struct RSTRPC_restorable_obj_root {
1                enum RSTRPC_ObjectLevel          objLevel;
1                int     backupApp;
1                STRING  (objName);
1                STRING  (objTypeString);
1        };

1        /* top level object data: in linked list,
                                        or in restorable object conatiner */

1        struct RSTRPC_top_level_obj {
1                struct RSTRPC_restorable_obj_root root;
1                OPAQUE  (appData);
1               STRING   (hostname);
1               STRING   (fileSpec);
1               STRING   (templateName);
1               STRING   (wiBIC);
1               char     wiType;
1               unsigned int flags;
1               RSTRPC_bool ssThread;    /* ?? is this app-specific? */
1        };

1        struct RSTRPC_tlo_list {
1                struct RSTRPC_tlo_list          *next;
1                struct RSTRPC_top_level_obj     *tlo;
1        };

1        struct RSTRPC_user_restorable_object
1        {
1                struct RSTRPC_restorable_obj_root root;
1                OPAQUE                  (appData);
1                u_long                  objMode;
1                STRING                  (objOwnerName);
1                STRING                  (objGroupName);
1                RSTRPC_time_ty          objModTime;
                                         /* Time obj was last modified */
1                struct RSTRPC_u_hyper   objSize;
1                enum RSTRPC_BackupStatus objBackupStatus;
1                STRING                  (objBaseName);
                                         /* Base name of the object */
1        };

1        struct RSTRPC_uro_list {
1                struct RSTRPC_uro_list                  *next;
1                struct RSTRPC_user_restorable_object    *uro;
1        };

1        struct RSTRPC_found_obj_list {
1                struct RSTRPC_found_obj_list            *next;
1                RSTRPC_time_ty                          time;
1                struct RSTRPC_user_restorable_object    *foundObj;
1        };




1        /*
1         * mediaObject definition
1         *
1         * The media object is an internal object of the Restore API.
```



```
                                                        It is presented
1         * to the users of the Restore API as an opaque object,
                                                        meaning that its content
1         * and format are not known except the Restore API internals.
                                                        It is used to
1         * inquire about the media that must be read to restore a set of
                                                        marked files.
1         */
1        struct RSTRPC_media_list {
1                struct RSTRPC_media_list        *next;
1                struct RSTRPC_media_object      *media_obj;
1        };

1        struct RSTRPC_media_object
1        {
1          STRING            (trail);          /* name of trail      */
1          STRING            (mtype);
                                    /* description of media type          */
1          STRING            (mtype_token);
                                    /* Simple media name: 8mm, DLT, EO    */
1          STRING            (barcode_label);
                                    /* ptr to the name of the barcode     */
1          STRING            (physical_loc);
                                    /* physical location of volume        */
1          STRING            (comments);
                                    /* user comments on the volume        */
1          STRING            (volid_ascii);
                                    /* the volume id for display          */
1          u_int             seqno;
                                    /* sequence # of volume               */
1          u_char            side;
                                    /* side of volume 'a' or 'b' or '\0'  */
1          RSTRPC_time_ty    lmtime;
                                    /* last modify time                   */
1          RSTRPC_bool       online;
                                    /* if TRUE then volume is online      */
1          RSTRPC_bool       offsite;
                                    /* if TRUE, then volume is offsite    */
1          RSTRPC_bool       is_orig;
                                    /* is this an original volume         */
1          RSTRPC_bool       run_media_dup;
                                    /* Does this trail run media dup      */
1          struct RSTRPC_media_list  *dups;  /* The list of duplicates */
1          short             num_dups;     /* The number of duplicates */
1          STRING            (luname);
1        };


         #endif   /* #ifndef _RESTORE_ENGINE_H_RPCGEN */

         #endif   /* H_RESTORE_RPC_DATA */
```

D3

```
%/*
%** Copyright 1997,1998 EMC Corporation
%*/

/*
**  Leading % causes rpcgen to pass a line directly thought to the output,
**  ie restore_engine.h in this case.  This allows the .h to make a little
**  more sense and be properly documented.
*/

%/*
% * restore_engine.x : EDM Restore Engine C/S communication module
% *
% * Mission Statement:  This is an RPCGEN file which defines the RPC interface
% *                     between the Restore Engine server (which resides on
% *                     the EDM server) and  the backup client callers of its
% *                     functions.  This defines the RPC level calls that a
% *                     "caller" can make and the "service" will respond to.
% *
% * Primary Data Acted On: This defines the data that will flow over the wire.
% *                        The RPC mechanism will take care of data
                                                              marshalling
% *
% *
% * Compile-Time Options:
% *                     This acutally gets run through RPCGEN not compiled.
                                                                    It
% *                      must be run through with the -h flag to create a
% *                      header, the -m flag to create the service side
% *                      routines, the -l flag to create the client side
% *                      routines, and the -c flag to create the common data
% *                      marshalling routines.
% *
% * Basic idea here:
% *                     Define the RPC level interfaces to the Restore Engine
% *                     and all data types that will be passed via RPC.
%*/

/* for sharing of STRING(x) and OPAQUE(x) */
#define  IN_DOTX
#include <restore/restoreRPC.h>

%#include <restore/dispatch_daemon.h>

/*****************************************************************************
 Constant Definitions
*****************************************************************************/

/*****************************************************************************
 Enum Definitions
*****************************************************************************/

/*****************************************************************************
 Typedef Definitions
*****************************************************************************/

typedef int      RE_errno_ty;

/*****************************************************************************
 Data Structure Definitions
*****************************************************************************/

/* Structure to start every RPC request and response - for debug purposes */
        struct RE_rpc_objID
        {
                unsigned long   rpc_type;
```



```
                                        /* RPC Object ID (ie, rpc #) */
                RSTRPC_time_ty  time;   /* creation time */
                long            len;
                                        /* Length of structure, version num? */
        };

        struct RE_null_args {
                RE_rpc_objID RPCobjID;
        };

        struct RE_status_result {
                RE_rpc_objID RPCobjID;
                RE_errno_ty status;
        };

        struct RE_boolean_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_bool     boolResult;
        };

        union RE_restorable_obj switch (RSTRPC_ObjectLevel objLevel)
        {
        case RSTRPC_tlo_type:
                RSTRPC_top_level_obj            *tloInfo;
        default:  /* anything else means NOT tlo -- i.e. container or leaf */
                RSTRPC_user_restorable_object   *uroInfo;
        } ;

const MAX_CHOICE_TEXT=80;

struct Choices {
    bool        isset;
    string      ctext<>;
    Choices     *nextchoice;
};

/* Question types */
const QTYPE_BOOL =      1;
const QTYPE_RAD =       2;
const QTYPE_MULTI =     4;
const QTYPE_STR =       8;
const QTYPE_YESNO =     16;
const QTYPE_INT =       32;

struct Question {
    int         qnum;
    int         qtype;
    int         maxlen;
    int         minlen;
    int         numchoices;
    string      invalidchars<>;
    string      headertxt<>;
    string      qtext<>;
    Choices     *choices;
};

struct Answer {
    int         qnum;
    string      ctext<>;
    Answer      *nextanswer;
};

struct AnswerList {
    int         numanswers;
    Answer      *firstanswer;
```

```
};


/* structures for input and output of re_initialize rpc call:  */

        struct RE_initialize_args {
                RE_rpc_objID RPCobjID;
                string username<>;
        };

/* structures for input and output of get_source_hosts and
 * get_destination_hosts rpc calls:
 */
        struct RE_get_hosts_args {
                RE_rpc_objID    RPCobjID;
                string          hostname<>;
                                        /* only for get_source_hosts */
                short           maxEntries;
                long            cookie;
        };

        struct RE_get_hosts_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                short           numEntries;     /* redundant but useful ? */
                long            cookie;
                RSTRPC_name_list        *hosts;  /* link to first hostname */
        };

/* structure for single character string argument */
        struct RE_string_args {
                RE_rpc_objID    RPCobjID;
                string  name<>;
        };

/* structure for GetHostPlatformType results */
        struct RE_get_host_platform_type_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                int             ptype;
        };

/* structures for input and output of submit RPCs
 */
        struct RE_submit_args {
                RE_rpc_objID    RPCobjID;
                string          hostname<>;
                string          directory<>;
                int             overwritePolicy;
                bool            inPlace;
                int             transport;
                int             submitObjectID;
                int             socketPort;
                string          socketClientName<>;
                string                  mapFile_env<>;
        };

        struct RE_get_submit_results_args {
                RE_rpc_objID    RPCobjID;
                bool            interrupt;
        };

        struct RE_get_submit_results_output {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                int             submitObjectID;
```



```
                                        /* handle for submit object */
                u_long          objectsDone;
        };

/* structures for input of Start RPC
 */
        struct RE_start_args {
                RE_rpc_objID    RPCobjID;
                int             submitObjectID;
                                        /* handle for submit object */
        };

/* structures for input and output of get_restore_feedback RPC
 */
        struct RE_get_restore_feedback_args {
                RE_rpc_objID    RPCobjID;
                bool            quit_restore;   /* flag to request cancel */
        };

        struct RE_Notification {
                int             msgtype;
                int             sourcemodule;
                int             level;
                int             msglen;
                string          msgtext<>;
                RE_Notification         *next;
        };

        struct RE_get_restore_feedback_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
               EPMStats         rstStats;
               RE_Notification  *notify;
        };

/* structure for output of get_question RPC
 */
        struct RE_get_question_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                Question        *query;
        };

/* structure for input of set_user_answer RPC
 */
        struct RE_set_user_answer_args {
                RE_rpc_objID    RPCobjID;
                AnswerList      answers;
        };

/* structures for input and output of get_top_level_objects RPC
 */
        struct RE_get_top_level_objects_args {
                RE_rpc_objID    RPCobjID;
                string          sourceHost<>;
                short           maxEntries;
                long            cookie;
        };

        struct RE_get_top_level_objects_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_tlo_list *topLevelObjs;  /* linked list */
                short           numEntries;
                long            cookie;
        };
```

```
/* structures for input and output of get_workitem_templates rpc call:
 */
        struct RS_get_top_level_templates_args {
                RE_rpc_objID    RPCobjID;
                RSTRPC_top_level_obj *topLevelObj;
                short           maxEntries;
                long            cookie;
        };

        struct RE_get_top_level_templates_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                short           numEntries;     /* redundant but useful ? */
                long            cookie;
                RSTRPC_name_list *templates;    /* link to first template */
        };

/* structure for input of does_alternate_exist rpc call:
 */
        struct RE_does_alternate_exist_args {
                RE_rpc_objID    RPCobjID;
                RSTRPC_top_level_obj *topLevelObj;
                string          templateName<>;
        };

/* structures for input and output of get_restorable_objects RPC's:
 */
        struct RE_get_restorable_objects_start_args {
                RE_rpc_objID    RPCobjID;
                RE_restorable_obj *parentObj;
                long            cookie;
                short           maxEntries;
                bool            allowBadFiles;
        };

        struct RE_get_restorable_objects_start_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
        };

        struct RE_get_restorable_objects_output_args {
                RE_rpc_objID    RPCobjID;
                short           maxEntries;
        };

        struct RE_get_restorable_objects_output_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_uro_list *childrenObjs;  /* linked list */
                long            numEntries;
                long            cookie;
        };

/* structures for input and output of find_restorable_objects RPC's:
 */
        struct RE_search_criteria {
                string  startDirectory<256>; /* Dir to start searching */
                bool    descendDirectory; /* Flag to descend into subdirs */
                string  searchString<128>;   /* String to search for */
                bool    excludeString;  /* Flag to include or exclude */
                RSTRPC_enum_ty  typeOfFile;
                                        /* Types of files to search for */
                string  owner<64>;      /* Specific owner of files */
                bool    excludeOwner;   /* Flag to exclude owner */
                string  group<64>;      /* Specific group of files */
```



```
                bool    excludeGroup;   /* Flag to exclude group */
                RSTRPC_u_hyper sizeInBytes;
                                        /* Specific size of files to find */
                RSTRPC_enum_ty  sizeMatch;
                                        /* type of matching to do for size */
                RSTRPC_time_ty  startTime;
                                        /* First backup date to use */
                RSTRPC_time_ty  endTime;        /* Last backup date to use */
        };

        struct RE_find_restorable_objects_args {
                RE_rpc_objID    RPCobjID;
                RE_search_criteria *searchCriteria;
        };

        struct RE_find_restorable_objects_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
        };

        struct RE_get_find_results_args {
                RE_rpc_objID    RPCobjID;
                bool            interrupt;      /* flag to request cancel */
                short           maxEntries;
                long            cookie;
        };

        struct RE_get_find_results_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_found_obj_list   *foundObjs;     /* linked list */
                long            numEntries;
                long            cookie;
        };

/* structures for input and output of mark_object RPC's:
 */
        struct RE_mark_object_args {
                RE_rpc_objID    RPCobjID;
                RSTRPC_user_restorable_object *thisObj;
                RSTRPC_time_ty  backupTime;
                bool            allowBadFiles;
                bool            descend;
        };

        struct RE_mark_object_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
        };

        struct RE_get_mark_results_args {
                RE_rpc_objID    RPCobjID;
                bool            interrupt;      /* flag to request cancel */
        };

        struct RE_get_mark_results_result {
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                u_long          badFileCount;
                u_long          permDenyFileCount;
                u_long          dirMarkCount;
                u_long          fileMarkCount;
                u_long          otherMarkCount;
        };

/* structures for input and output of unmark_object RPC's:
```

```
 */
        struct RE_unmark_object_args {
                RE_rpc_objID     RPCobjID;
                RSTRPC_user_restorable_object *thisObj;
                RSTRPC_time_ty   backupTime;
                bool             badFilesOnly;
                bool             descend;
        };

        struct RE_get_unmark_results_result {
                RE_rpc_objID     RPCobjID;
                RE_errno_ty      status;
                u_long           badFileCount;
                u_long           dirMarkCount;
                u_long           fileMarkCount;
                u_long           otherMarkCount;
        };

/* structure for output of get_marked_total_size RPC:
 */
        struct RE_get_marked_total_size_result {
                RE_rpc_objID     RPCobjID;
                RE_errno_ty      status;
                RSTRPC_u_hyper   total;
        };

/* structure for output of get_current_template RPC:
 */
        struct RE_get_current_template_result {
                RE_rpc_objID     RPCobjID;
                RE_errno_ty      status;
                string           templateName<>;
                RSTRPC_bool      alternate;
        };

/* structure for output of get_current_backup_time RPC:
 */
        struct RE_get_current_backup_time_result {
                RE_rpc_objID     RPCobjID;
                RE_errno_ty      status;
                RSTRPC_time_ty   backupTime;
        };

/* structure for input and output of get_all_backup_times RPC:
 */
        struct RE_get_all_backup_times_args {
                RE_rpc_objID     RPCobjID;
                RSTRPC_time_ty   startTime;
                RSTRPC_time_ty   endTime;
                RSTRPC_backup_flags_ty flags;
                long             maxEntries;
                long             cookie;
        };

        struct RE_get_all_backup_times_result {
                RE_rpc_objID     RPCobjID;
                RE_errno_ty      status;
                RSTRPC_time_list *backupTimes;
                long             numEntries;
                long             cookie;
        };

/* structure for input of is_there_xxxx_backup_for_time and
                                                    set_backup_for_time
 * RPC's:
 */
```



```
        struct RE_backup_for_time_args {
                RE_rpc_objID     RPCobjID;
                RSTRPC_time_ty   time;
                RSTRPC_backup_flags_ty flags;
        };

/* structure for input of set_'relative'backup * RPC's: */
        struct RE_set_backup_time_args {
                RE_rpc_objID     RPCobjID;
                RSTRPC_backup_flags_ty flags;
        };

/* structure for input and output of get_necessary_media RPC:
 */
        struct RE_get_necessary_media_args {
                RE_rpc_objID     RPCobjID;
                long             maxEntries;
                RSTRPC_bool      all;
                long             cookie;
        };

        struct RE_get_necessary_media_result {
                RE_rpc_objID     RPCobjID;
                RE_errno_ty      status;
                RSTRPC_media_list *mediaList;
                short            numEntries;
                long             cookie;
        };

/* structures for input and output of is_object_markable RPC:
 */
        struct RE_is_object_markable_args {
                RE_rpc_objID             RPCobjID;
                RSTRPC_user_restorable_object   *thisObject;
        };

        struct RE_is_object_markable_result {
                RE_rpc_objID     RPCobjID;
                RE_errno_ty      status;
                bool             markable;
        };

/* structures for input and output of is_object_marked RPC:
 */
        struct RE_is_object_marked_args {
                RE_rpc_objID     RPCobjID;
                RSTRPC_uro_list *objList;
                u_long           numEntries;
        };

        struct RE_is_object_marked_result {
                RE_rpc_objID     RPCobjID;
                RE_errno_ty      status;
                u_long           numMarked;
                RSTRPC_bool      marked<>;
        };

/* structures for input and output of is_object_searchable and
 * get_backup_times_support RPCs:
 */
        struct RE_tlo_query_args {
                RE_rpc_objID             RPCobjID;
                RSTRPC_top_level_obj     *topLevelObj;
        };

        struct RE_catalog_info {
```

```
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                string          level<>;
                string          numrec<>;
                string          catType<>;
        };


/* structure for inputs that require only time
 */
        struct RE_time{
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_time_ty  backupTime;
        };


        struct RE_recx_file_info{
                RE_rpc_objID    RPCobjID;
                RE_errno_ty     status;
                RSTRPC_recx_file_info fileinfo;
        };

program EDM_RESTORE_ENGINE {

   version EDMRE_FUNCTIONS {

            /* rpc for EDMRST_Initialize */
            RE_status_result
                re_initialize( RE_initialize_args ) = 1;

            /* rpc for EDMRST_GetSourceHosts */
            RE_get_hosts_result
                re_get_source_hosts( RE_get_hosts_args ) = 2;

            /* rpc for EDMRST_GetTopLevelObjects */
            RE_get_top_level_objects_result
                re_get_top_level_objects( RE_get_top_level_objects_args ) = 3;

            /* rpc for EDMRST_GetTopLevelTemplates */
            RE_get_top_level_templates_result
                re_get_top_level_templates(
                                        RE_get_top_level_templates_args ) = 4;

            /* rpc for EDMRST_Submit */
            RE_status_result
                re_submit( RE_submit_args ) = 5;

            /* rpc for EDMRST_GetSubmitResults */
            RE_get_submit_results_output
                re_get_submit_results( RE_get_submit_results_args ) = 6;

            /* rpc for EDMRST_Start */
            RE_status_result
                re_start( RE_start_args ) = 7;

            /* rpc for EDMRST_GetRestoreFeedback */
            RE_get_restore_feedback_result
                re_get_restore_feedback( RE_get_restore_feedback_args ) = 8;

            /* rpc for EDMRST_GetQuestion */
            RE_get_question_result
                re_get_question( RE_null_args ) = 9;

            /* rpc for EDMRST_SetUserAnswer */
            RE_status_result
```



```
                re_set_user_answer( RE_set_user_answer_args ) = 10;

            /* rpc for EDMRST_Finish */
            RE_status_result
                re_finish( RE_null_args ) = 11;

            /* rpc for EDMRST_DoesAlternateExist */
            RE_boolean_result
                re_does_alternate_exist( RE_does_alternate_exist_args ) = 12;
            /* rpc's for EDMRST_GetRestorableObjects */
            RE_get_restorable_objects_start_result
                re_get_restorable_objects_start(
                                RE_get_restorable_objects_start_args ) = 13;
            RE_get_restorable_objects_output_result
                re_get_restorable_objects_output(
                                RE_get_restorable_objects_output_args ) = 14;

            /* rpc's for EDMRST_FindRestorableObjects */
            RE_find_restorable_objects_result
                re_find_restorable_objects(
                                RE_find_restorable_objects_args ) = 15;
            RE_get_find_results_result
                re_get_find_results( RE_get_find_results_args ) = 16;

            /* rpc's for EDMRST_MarkObject */
            RE_mark_object_result
                re_mark_object( RE_mark_object_args ) = 17;
            RE_get_mark_results_result
                re_get_mark_results( RE_get_mark_results_args ) = 18;

            /* rpc's for EDMRST_UnmarkObject */
            RE_mark_object_result
                re_unmark_object( RE_unmark_object_args ) = 19;
            RE_get_unmark_results_result
                re_get_unmark_results( RE_get_mark_results_args ) = 20;

            /* rpc for EDMRST_GetMarkedTotalSize */
            RE_get_marked_total_size_result
                re_get_marked_total_size( RE_null_args ) = 21;

            /* rpc for EDMRST_GetCurrentTemplate */
            RE_get_current_template_result
                re_get_current_template( RE_null_args ) = 22;

            /* rpc for EDMRST_GetCurrentBackupTime */
             RE_get_current_backup_time_result
                re_get_current_backup_time( RE_null_args ) = 23;

            /* rpc for EDMRST_GetAllBackupTimes */
             RE_get_all_backup_times_result
                re_get_all_backup_times( RE_get_all_backup_times_args ) = 24;

            /* rpc for EDMRST_IsTherePrevBackup */
             RE_boolean_result
                re_is_there_prev_backup( RE_set_backup_time_args ) = 25;

            /* rpc for EDMRST_IsThereNextBackup */
             RE_boolean_result
                re_is_there_next_backup( RE_set_backup_time_args ) = 26;

            /* rpc for EDMRST_IsTherePrevBackupForTime */
             RE_boolean_result
                re_is_there_prev_backup_for_time(
                                        RE_backup_for_time_args) = 27;

            /* rpc for EDMRST_IsThereNextBackupForTime */
```

```
    RE_boolean_result
        re_is_there_next_backup_for_time(
                                        RE_backup_for_time_args) = 28;

    /* rpc for EDMRST_SetBackupForTime */
    RE_status_result
        re_set_backup_for_time( RE_backup_for_time_args ) = 29;

    /* rpc for EDMRST_SetPrevBackup */
    RE_status_result
        re_set_prev_backup( RE_set_backup_time_args ) = 30;

    /* rpc for EDMRST_SetNextBackup */
    RE_status_result
        re_set_next_backup( RE_set_backup_time_args ) = 31;

    /* rpc for EDMRST_SetFirstBackup */
    RE_status_result
        re_set_first_backup( RE_set_backup_time_args ) = 32;

    /* rpc for EDMRST_SetMostRecentBackup */
    RE_status_result
        re_set_most_recent_backup( RE_set_backup_time_args ) = 33;

    /* rpc for EDMRST_GetNecessaryMedia */
    RE_get_necessary_media_result
        re_get_necessary_media( RE_get_necessary_media_args ) = 34;

    /* rpc for EDMRST_IsObjectMarkable */
    RE_is_object_markable_result
        re_is_object_markable( RE_is_object_markable_args ) = 35;

    /* rpc for EDMRST_IsObjectMarked */
    RE_is_object_marked_result
        re_is_object_marked( RE_is_object_marked_args ) = 36;

    /* rpc for EDMRST_GetDestinationHosts */
    RE_get_hosts_result
        re_get_destination_hosts( RE_get_hosts_args ) = 37;

    /* rpc for EDMRST_GetHostPlatformType */
    RE_get_host_platform_type_result
        re_get_host_platform_type( RE_string_args ) = 38;

    /* rpc for EDMRST_IsObjectSearchable */
    RE_boolean_result
        re_is_object_searchable( RE_tlo_query_args ) = 39;

    /* rpc for EDMRST_GetBackupTimesSupport */
    RE_boolean_result
        re_get_backup_times_support( RE_tlo_query_args ) = 40;

    /* rpc for EDMRE_Load_recx_directives*/
    RE_status_result
        re_load_recx_directives( RE_recx_file_info ) = 41;

    /* rpc for EDMRST_poll_Load_recx_directives */
    RE_status_result
        re_poll_load_recx_directives(RE_null_args ) = 42;

    /* rpc for RSTSL_get_backup_level */
    RE_catalog_info
        re_get_catalog_info(RE_time) = 43;

    /* rpc for EDMRST_GetAllTopLevelObjects */
    RE_get_top_level_objects_result
```



```
        re_get_all_top_level_objects(
                                        RE_get_top_level_objects_args ) = 44;

    /* rpc for EDMRST_GetSymmRestoreOption */
    RE_boolean_result
        re_get_symm_restore_option( RE_tlo_query_args ) = 45;

    /* rpc for EDMRST_Ping */
    RE_status_result
        re_ping( RE_null_args ) = 46;


} = 1; /* This is version 1 */

%/* This is the RPC program number.
                                        These are reserved in /pds/docs/RPC_numbers
% * This number cannot be re-used by any other RPC daemon on the machine,
                                                                    as it
% * identifies this daemon uniquely.  If it were to be re-used,
                                                                the last daemon
% * to register would be contacted when RPC's come in for this number.
%*/
} = 390016;
```

D4

```
/*****************************************************************************
                                                                      ******
**
** File Name:   RSTgettlob.c
**
** Copyright (c) 1998,1999 by EMC Corporation.
**
** Purpose:
**       This module contains the EDMRST_GetTopLevelObjects
**       Restore API function.
**       This function is provided to allow retrieval of the
**       top level objects which are restorable for the given client.
**
**
** Compile-Time Options:
**              This section must list any compile time definitions
**              which will affect this header.
**
*****************************************************************************
                                                                      *****/



/* The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
*/

#ifndef lint
static char RCS_id [] = "$RCSfile$ "
                        "$Revision$ "
                        "$Date$" ;
#endif


/*
 * Feature test switches.
 * Standard defines required to turn on OS features go here.
 *
 * The following is required for code that uses POSIX API's.
 * Remove for non-POSIX, non-portable code.
 */

#define  _POSIX_SOURCE 1


/*
 * System headers.
 */


/*
 * Epoch headers.
 */
#include <eb/eb_port.h>
#include <eb/rb_log.h>


/*
 * Local headers
 */
#include <RSTinterns.h>
#include <RSTsup_rpc.h>
#include <RSTsup_csm.h>


/*
```



```
 * External declarations
 */
```

```
 69  /*****************************************************************
 70   * EDMRST_GetTopLevelObjects:
 71   *
 72   *  This function is provided to allow retrieval of the
 73   *  work items which are restorable for the given client.
 74   *
 75   *  It is a GOAL of this routine to return all work-items ever backed
 76   *  up successfully.  Currently, though, it only looks in the config
 77   *  file for work-items of the given client.
 78   *
 79   *  The cookie must be initialized to INIT_COOKIE on the first call to
                                                                    this
 80   *  routine.
 81   *      This routine will update the cookie to allow retrieval of more
 82   *  objects if there are more than "maxEntries".  The cookie will be
 83   *  returned as DONE_COOKIE when there are no more to retrieve.
 84   *
 85   *  Parameters:
 86   *    svrHdl          (
 87   *                     I)  - A pointer to this user's client handle for the
 88   *                           Restore Engine (server) connection.
 89   *    sourceHost      (I)  - the name of the source host being restored
 90   *    maxEntries      (I)  - the maximum number of objects to return
 91   *    topLevObjs      (
 92   *                     O)  - ptr to pre-allocated array of restorableObject
 93   *                           buffer ptrs
 94   *    numberEntries   (
 95   *                     O)  - the real number of objects returned in the array
 96   *    cookie          (IO) - a place holder for the list position
 97   *                           meaningful to only the internals of the API
 98   *
 99   *****************************************************************/
100
101  eerrno_ty
102  EDMRST_GetTopLevelObjects( serverHandle         svrHdl,
103                             const char           *sourceHost,
104                             const short          maxEntries,
105                             restorableObjectPtr *topLevObjs,
106                             short               *numberEntries,
107                             long                *cookie )
108  1 {
109  1       RE_get_top_level_objects_result         *rpc_result;
110  1       RE_get_top_level_objects_args   rpc_args;
111  1       RSTRPC_tlo_list                                 *temp_list;
112  1       eerrno_ty                               result = E_SUCCESS ;
113  1       short                                   index;
114  1       restorableObject                **objPtrArray = (
115                                                  restorableObject **)topLevObjs;
116
117  1       rbe_log_debug_sub( 0, "EDMRST_GetTopLevelObjects called" );
118
119  1       /* validate args first: */
120  1       if (sourceHost==NULL
121  1         || numberEntries==NULL
122  1         || cookie==NULL
123  1         || maxEntries <= 0
124  1         || topLevObjs== NULL
125  1         || svrHdl==NULL )
126  1               return( EP_RB_RECOVER_BAD_ARGS );
127
128  1       /* validate target restorableObjects: */
129  1       for ( index=0; index<maxEntries; index++ )
130  2       {
131  2               if (
132                 RESTORABLE_OBJECT != objPtrArray[index]->restoreObjType
133                   || NULL != objPtrArray[index]->rpcObjPtr )
```



```
128  2                  return( EP_RB_RECOVER_BAD_ARGS );
129  1          }
130  1          /* Prepare input argument structure for RPC: */
131  1          rpc_args.sourceHost = (char *)sourceHost;
132  1          rpc_args.maxEntries = maxEntries;
133  1          rpc_args.cookie = *cookie;
134  1          set_rpc_obj( re_get_top_level_objects, &rpc_args.RPCobjID );

136  1          rpc_result = re_get_top_level_objects_1( &rpc_args, svrHdl );
137  2          if (!rpc_result) {
138  2                  rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL);
139  2                  return( EP_RB_RECOVER_RPC_FAIL );
140  1          }

142  1          /* move results to caller's area, if successful: */
143  1          if (rpc_result->status == E_SUCCESS)
144  2          {
145  2                  *cookie = rpc_result->cookie;
146  2                  *numberEntries = rpc_result->numEntries;
147  2                  index = 0;
148  2                  while ( rpc_result->numEntries )
149  3                  {
150  3                          temp_list = rpc_result->topLevelObjs;
151  3                          if ( !temp_list || !rpc_args.maxEntries-- )
152  3                                  /* some null pointer or too many
                                                                  returned */
153  3                                  break;
154  3                          objPtrArray[index++]->rpcObjPtr
155  3                                  = (
                                    RSTRPC_restorable_obj_root *)temp_list->tlo;
156  3                          /* need this to end with NULL in
                                                          rpc_result->topLevelObjs,
157  3                           * because returned top level objects can't be freed
                                                                  yet */
158  3                          rpc_result->topLevelObjs = temp_list->next;
159  3                          free( temp_list );
160  3                          rpc_result->numEntries--;
161  2                  }
162  2                  if (rpc_result->numEntries)
163  2                          rpc_result->status = EP_RB_RECOVER_SERVERFAIL;
164  1          }

166  1          result = rpc_result->status;

168  1          /* release RPC result struct: */
169  1          xdr_free( xdr_RE_get_top_level_objects_result, (
                                                          char *)rpc_result );

171  1          return( result );
172      }      /* end of EDMRST_GetTopLevelObjects() */
```

```
176  /*****************************************************************************
177   * EDMRST_GetAllTopLevelObjects:
178   *
179   *  This function is provided to allow retrieval of the
180   *  work items which are restorable for the given client.
181   *
182   *  It is a GOAL of this routine to return all work-items ever backed
183   *  up successfully.  Currently, though, it only looks in the config
184   *  file for work-items of the given client.
185   *
186   *  The cookie must be initialized to INIT_COOKIE on the first call to
187   *                                                              this
      *  routine.
188   *      This routine will update the cookie to allow retrieval of more
189   *  objects if there are more than *maxEntries*.  The cookie will be
190   *  returned as DONE_COOKIE when there are no more to retrieve.
191   *
192   *  Parameters:
193   *    svrHdl          (
      *                    I)  - A pointer to this user's client handle for the
      *                                  Restore Engine (server) connection.
194   *    sourceHost      (I)  - the name of the source host being restored
195   *    maxEntries      (I)  - the maximum number of objects to return
196   *    topLevObjs      (
      *                    O)  - ptr to pre-allocated array of restorableObject
      *                                  buffer ptrs
197   *
198   *    numberEntries (
      *                   O)  - the real number of objects returned in the array
199   *    cookie          (IO) - a place holder for the list position
      *                                  meaningful to only the internals of the API
200   *
201   *
202   *****************************************************************************/

204  eerrno_ty
205  EDMRST_GetAllTopLevelObjects( serverHandle         svrHdl,
206                                const char          *sourceHost,
207                                 short              maxEntries,
208                                 restorableObjectPtr *topLevObjs,
209                                 short              *numberEntries,
210                                 long               *cookie )
211  1 {
212  1         RE_get_top_level_objects_result          *rpc_result;
213  1         RE_get_top_level_objects_args   rpc_args;
214  1         RSTRPC_tlo_list                          *temp_list;
215  1         eerrno_ty                        result = E_SUCCESS ;
216  1         short                            index;
217  1         restorableObject          **objPtrArray = (
                                                restorableObject **)topLevObjs;


219  1         rbe_log_debug_sub( 0, "EDMRST_GetAllTopLevelObjects called" );

221  1         /* validate args first: */
222  1         if (sourceHost==NULL
223  1           || numberEntries==NULL
224  1           || cookie==NULL
225  1           || maxEntries <= 0
226  1           || topLevObjs== NULL
227  1           || svrHdl==NULL )
228  1                   return( EP_RB_RECOVER_BAD_ARGS );


230  1         /* validate target restorableObjects: */
231  1         for ( index=0; index<maxEntries; index++ )
232  2         {
```



```
233  2                  if (
                              RESTORABLE_OBJECT != objPtrArray[index]->restoreObjType
234  2                           || NULL != objPtrArray[index]->rpcObjPtr )
235  2                        return( EP_RB_RECOVER_BAD_ARGS );
236  1         }
237  1         /* Prepare input argument structure for RPC: */
238  1         rpc_args.sourceHost = (char *)sourceHost;
239  1         rpc_args.maxEntries = maxEntries;
240  1         rpc_args.cookie = *cookie;
241  1         set_rpc_obj(
                              re_get_all_top_level_objects, &rpc_args.RPCobjID );


243  1         rpc_result = re_get_all_top_level_objects_1(
                                                  &rpc_args, svrHdl );
244  2         if (!rpc_result) {
245  2                   rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL);
246  2                   return( EP_RB_RECOVER_RPC_FAIL );
247  1         }


249  1         /* move results to caller's area, if successful: */
250  1         if (rpc_result->status == E_SUCCESS)
251  2         {
252  2                   *cookie = rpc_result->cookie;
253  2                   *numberEntries = rpc_result->numEntries;
254  2                   index = 0;
255  2                   while ( rpc_result->numEntries )
256  3                   {
257  3                             temp_list = rpc_result->topLevelObjs;
258  3                             if ( !temp_list || !rpc_args.maxEntries-- )
259  3                                       /* some null pointer or too many
                                                                  returned */
260  3                                       break;
261  3                             objPtrArray[index++]->rpcObjPtr
262  3                                       = (
                                        RSTRPC_restorable_obj_root *)temp_list->tlo;
263  3                             /* need this to end with NULL in
                                                       rpc_result->topLevelObjs,
264  3                              * because returned top level objects can't be freed
                                                                  yet */
265  3                             rpc_result->topLevelObjs = temp_list->next;
266  3                             free( temp_list );
267  3                             rpc_result->numEntries--;
268  2                   }
269  2                   if (rpc_result->numEntries)
270  2                             rpc_result->status = EP_RB_RECOVER_SERVERFAIL;
271  1         }

273  1         result = rpc_result->status;

275  1         /* release RPC result struct: */
276  1         xdr_free( xdr_RE_get_top_level_objects_result, (
                                                            char *)rpc_result );


278  1         return( result );
279  }         /* end of EDMRST_GetNetworkTopLevelObjects() */
```

Fri Oct 10 11:48:36 2008 Function Index Page 2

Fri Oct 10 11:48:36 2008 Function Index Page 1

Fri Oct 10 11:48:36 2008 Function Index Page 2

```
/*****************************************************************************
                                                                      ******
**
** File Name:   RSLgettlob.c
**
** Copyright (c) 1998,1999 by EMC Corporation.
**
** Purpose:
**       This module contains the RSTSL_GetTopLevelObjects
**       Restore Service Library function.
**       This function is provided to allow retrieval of the
**       top level objects which are restorable for the given client.
**
**
** Compile-Time Options:
**               This section must list any compile time definitions
**               which will affect this header.
**
*****************************************************************************
                                                                      *****/


/* The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
*/

#ifndef lint
static char RCS_id [] = "$RCSfile$ "
                        "$Revision$ "
                        "$Date$" ;
#endif


/*
 * Feature test switches.
 * Standard defines required to turn on OS features go here.
 *
 * The following is required for code that uses POSIX API's.
 * Remove for non-POSIX, non-portable code.
 */

#define  _POSIX_SOURCE 1


/*
 * System headers.
 */


/*
 * Epoch headers.
 */
#include <eb/eb_port.h>
#include <eb/rb_log.h>


/*
 * Local headers
 */
#include <RSLinterns.h>


/*
 * #defines, structures, typedefs local to this source file
 */
```



```
typedef struct wi_info
{
    struct wi_info *next;
    char *wi_name;
    char *wi_work;
    char *wi_bic;
    char wi_type;
} wi_info;

NEW_SRC_FILE();

/*
 * External declarations
 */
```

```
 82  /*****************************************************************
 83   * RSTSL_GetTopLevelObjects:
 84   *
 85   *  This function is provided to allow retrieval of the
 86   *  work items which are restorable for the given client.
 87   *
 88   *  It is a GOAL of this routine to return all work-items ever backed
 89   *  up successfully.  Currently, though, it only looks in the config
 90   *  file for network workitems of the given client.
 91   *
 92   *  The cookie must be initialized to lNIT_COOKIE on the first call to
                                                                    this
 93   *  routine.
 94   *      This routine will update the cookie to allow retrieval of more
 95   *  objects if there are more than "maxEntries".  The cookie will be
 96   *  returned as DONE_COOKIE when there are no more to retrieve.
 97   *
 98   *  Parameters:
 99   *    sourceHost      (I)   - the name of the source host being restored
100   *    maxEntries      (I)   - the maximum number of objects to return
101   *    topLevObjs      (O)   - ptr to linked list of Top Level Objects
102   *    numberEntries (
                    O)   - the real number of objects returned in the array
103   *    cookie          (IO) - a place holder for the list position
104   *                           meaningful to only the internals of the API
105   *
106   *****************************************************************/
107
108  eerrno_ty
109  RSTSL_GetTopLevelObjects( const char        *sourceHost,
110                            const short       maxEntries,
111                            struct RSTRPC_tlo_list **topLevObjs,
112                            short             *numberEntries,
113                            long              *cookie,
114  1                         int                execMode)
115  1 {
116  1     RBC_WORKGROUP *wgp;
117  1     short         index;
118  1     wi_info       *n_node;
119  1     wi_info       *tmp_list;
120  1     short         tmp_count;
121  1     eerrno_ty     result = E_SUCCESS;
122  1     void          *saved_appdata;
123  1     struct RSTRPC_tlo_list       *tloPtr;
124  1     struct RSTRPC_tlo_list       *tloLast;
125  1     struct pluginData            *piPtr;
126
127  1     /*
128  1      * Note that the following variables are declared static so that
129  1      * values set during the initial call to this routine will be
130  1      * of use on subsequent calls when no explicit set takes place.
131  1      */
132
133  1     static        short           wi_count;
134  1     static        short           total_count;
135  1     static         long           valid_cookie;
136  1     static        wi_info         *wi_list;
137  1     static        wi_info         *top_wi_list=NULL;
138  1     static         short          pi_count;
                                        /* num tlos left from plugins */
139  1     static        struct RSTRPC_tlo_list *pluginList = NULL;
140
141  1     /*
142  1      * Check to make sure rcp has the in memory config info.
143  1      */
```



```
144  1     * If the pointer is NULL, return error.
145  1     */

147  1    if (NULL == rcp->rc_config)
148  2    {
149  3        return(EP_RB_RECOVER_BAD_CONTEXT);
150  1    }

152  1    /*
153  1     * Check value of cookie.  If it is the init cookie, then call
154  1     * rb_get_auth_clients() otherwise, feed off of list of names that
155  1     * were generated by the first call.
156  1     */

158  1    if (NULL == cookie)
159  2    {
160  3        return(EP_RB_RECOVER_BAD_COOKIE);
161  1    }

163  1    /*
164  1     * Fill in the recover context source client hostname field
165  1     * if it currently is set to something, free up that memory first.
166  1     */

168  1    if (NULL != rcp->rc_source_client_hostname)
169  2    {
170  2        free(rcp->rc_source_client_hostname);
171  1    }

173  1    if (sourceHost && *sourceHost != 0)
174  2    {
175  2        rcp->rc_source_client_hostname = esl_strdup(sourceHost);
176  2        if (NULL == rcp->rc_source_client_hostname)
177  3        {
178  3            rec_api_log_csm(SUB_CSM_NOMEM, NULL);
179  3            return(EP_RB_RECOVER_NOMEM);
180  2        }
181  1    }
182  1    else
183  2    {
184  2        return EP_RB_RECOVER_BAD_ARGS;
185  1    }

187  1    rbe_log_debug(
                0, "In GetTLO for %s, cookie = %x", sourceHost, *cookie);

189  1    if (INIT_COOKIE == *cookie)
190  2    {
191  2        /*
192  2         * Set initial values since the cookie indicates the INIT
                                                                    state.
193  2         */

195  2        wi_count = 0;
196  2        pi_count = 0;
197  2        wi_list = NULL;

199  2        /*
200  2         * Check to see if there is already an in-memory static list
                                                                    of
201  2         * workitems being used by this function.  If so,
                                                        free this list
202  2         * since the caller has reset the cookie to INIT without
                                                                  having
203  2         * exhausted the list.
204  2         */
```

```
206 2           while (NULL != top_wi_list)
207 3           {
208 3               if (NULL != top_wi_list->wi_name)
209 4               {
210 4                   free(top_wi_list->wi_name);
211 3               }
212 3               if (NULL != top_wi_list->wi_work)
213 4               {
214 4                   free(top_wi_list->wi_work);
215 3               }
216 3               if (NULL != top_wi_list->wi_bic)
217 4               {
218 4                   free(top_wi_list->wi_bic);
219 3               }
220 3               tmp_list = top_wi_list->next;
221 3               free(top_wi_list);
222 3               top_wi_list = tmp_list;
223 2           }

225 2   /* Free unused top level objects from plugins */
226 2           if (pluginList)
227 3           {
228 3                   RSTSL_FreeRestorableObjectList( pluginList );
229 3                   pluginList = NULL;
230 2           }

232 2           /*
233 2            * Scan all the workitems in all the workgroups
234 2            * We are going to get all of the items that meet the
                                                              criteria
235 2            * and keep them in static memory.
                                                  Further calls will feed off
236 2            * of this list.
237 2            */

239 2           for (wgp = rcp->rc_config->pgrouplist;
240 2                wgp != NULL ;
241 2                wgp = wgp->next)
242 3           {
243 3               RBC_WORKITEM *wip;

245 3               for (wip = wgp->pwilist; wip != NULL; wip = wip->next)
246 4               {
247 4                   /* host must match and witype must not be a plug-in's
                                                                        */
248 4                   if ( (0 == strcmp(wip->sysname, sourceHost))
249 4                     && ((execMode == RAW_NETWORK)
250 4                       || (0 == rcp->rc_num_plugin_wi_types
251 4                       || NULL == memchr(
                                    rcp->rc_plugin_wi_types, wip->wi_type,
252 4                                          rcp->rc_num_plugin_wi_types )
                                                                    ) ) )

253 5                   {
254 5                       n_node = linked_list_new((
                              generic_list_ty **) &wi_list, sizeof(wi_info));
255 6                       if (n_node == NULL) {
256 6                           result = EP_RB_RECOVER_NOMEM;
257 6                           break;
258 5                       }
259 5                       n_node->wi_name = esl_strdup(wip->name);
260 5                       if (NULL == n_node->wi_name)
261 6                       {
262 6                           result = EP_RB_RECOVER_NOMEM;
263 6                           break;
264 5                       }
```



```
266 5                   /*
267 5                    * For striped workitems, the work item list can be
268 5                    * NULL for the stripes > 1 of workitems without a
269 5                    * partitionspec.
270 5                    */
271 5                       if (
                        NULL != wip->list)      /* else wi_work already NULL */
272 6                       {
273 6                           n_node->wi_work = esl_strdup(wip->list);
274 6                           if (NULL == n_node->wi_work)
275 7                           {
276 7                               result = EP_RB_RECOVER_NOMEM;
277 7                               break;
278 6                           }
279 5                       }

281 5                       if (NULL != wip->backup_init)
282 6                       {
283 6                           n_node->wi_bic = esl_strdup(wip->backup_init);
284 6                           if (NULL == n_node->wi_bic)
285 7                           {
286 7                               result = EP_RB_RECOVER_NOMEM;
287 7                               break;
288 6                           }
289 5                       }

291 5                       n_node->wi_type = wip->wi_type;

293 5                       ++wi_count;
294 4                   }
295 3               } /* end inner for(
                                  ) scanning workitems within a workgroup */

297 3               if (
                      result != E_SUCCESS)    /* malloc failure, break out */
298 3                   break;

300 2           } /* end outer for() scanning workgroups */

302 2           total_count = 1;  /* Needs to be 1 based */
303 2           valid_cookie = INIT_COOKIE; /* Clean up cookie crumbs */

305 2           /*
306 2            * Save the pointer to the head of the list for freeing later.
307 2            */

309 2           top_wi_list = wi_list;

311 3           if (result != E_SUCCESS) {
312 4                   if (result == EP_RB_RECOVER_NOMEM) {
313 4                       rec_api_log_csm(SUB_CSM_NOMEM, NULL);
314 3                   }
315 3                   /* free wi_list next time in */
316 3                   return result;
317 2           }

319 2           if (execMode != RAW_NETWORK)
320 3               {
321 3               /* save appData pointer in case a tlo is selected already */
322 3               saved_appdata = rcp->appData;

324 3               /* get top level object list(s) from plugin(s) */
325 3               for ( index = 1, piPtr = rcp->piList;
326 3                     NULL != piPtr;
327 3                     index++, piPtr = piPtr->next )
```

```
328 4                   {
329 4                     tloPtr = NULL;
330 4                     tmp_count = 0;
331 4                     rcp->appData = piPtr->appData;
332 4                     result = piPtr->piFuncArrey[PIFuncIndexGetTLO]
333 4                                 (
                                        rcp, sourceHost, &tloPtr, &tmp_count );
334 4                     if ( result != E_SUCCESS
335 4                       || (NULL == tloPtr && tmp_count > 0)
336 4                       || (NULL != tloPtr && tmp_count == 0) )
337 5                     { /* error or inconsistent results */
338 5                             rbe_internal_error(
339 5                                     result,
340 5                                     "plugin: %s;
                                          error in GetTopLevelObjects call",
341 5                                     ((struct pluginIDdata *)(
                                                piPtr->idData))->name
342 5                             );
343 5
344 5                             result = E_SUCCESS;      /* not fatal */
345 5                             continue;
                                                /* try next plugin anyway */
346 4                     }
347 4                     if (NULL == tloPtr)
348 4                             continue;        /* none from this plug-in */
349 4
350 4                     if (NULL == pluginList)
351 4                             pluginList = tloPtr;
352 4                     else
353 4                             tloLast->next = tloPtr;
354 4
355 4                     for ( ; tloPtr && tmp_count;
                                              tmp_count--, pi_count++ )
356 5                     {
357 5                             tloPtr->tlo->root.backupApp = index;
358 5                             tloLast = tloPtr;
359 5                             tloPtr = tloPtr->next;
360 4                     }
361 4
362 4                     if (
                        tmp_count || tloPtr)         /* inconsistent output */
363 5                     {
364 5                             rbe_internal_error(
365 5                                     0,
366 5                                     "plugin: %s;
                                          bad output from GetTopLevelObjects call",
367 5                                     ((struct pluginIDdata *)(
                                                piPtr->idData))->name
368 5                             );
369 4                     }
370 3             }
371 3
372 3             rcp->appData = saved_appdata;
                                              /* restore saved appData ptr */
373 2           }
374 2           else
375 3             {
376 3               pluginList = NULL;
377 3               pi_count = 0;
378 2             }
379 1     }
380 1     else if ((valid_cookie != *cookie) || (DONE_COOKIE == *cookie))
381 2     {
382 2           return(EP_RB_RECOVER_BAD_COOKIE);
383 1     }
```



```
385 1     /*
386 1      * Logic drops through to here regardless of the passed in cookie
387 1      * being the INIT state or a valid key in a subsequent call.
388 1      */
389 1
390 1     /*
391 1      * Return a linked list of the number of "restorable objects"
                                                          requested.
392 1      * Stop if we reach the end of the list.
393 1      */
394 1
395 1     /* set linked list and current entry pointers to NULL at start */
396 1     *topLevObjs = tloPtr = NULL;
397 1
398 1     index = 0;
399 1     while ( (total_count <= wi_count + pi_count)
400 1             && (index < maxEntries)
401 1             && ( (wi_list != NULL) || (pluginList != NULL) ) )
402 2     {
403 2          if (wi_list != NULL)
404 3          {
405 3               tloPtr = linked_list_new( (
                                              generic_list_ty **) topLevObjs,
                                               sizeof(
                                                    struct RSTRPC_tlo_list));
406 3
407 3               if ( (NULL == tloPtr)
408 3                 || ((tloPtr->tlo = calloc(1,sizeof(
                                                    struct RSTRPC_top_level_obj)) )
409 3                     == NULL) )
410 4                 {
411 4                 /* do we return partial list, or free list & return none ? */
412 4                 /* need a local function to free list of restorable objects
                                                                  on error */
413 4                         result = EP_RB_RECOVER_NOMEM;
414 4                         break;
415 3                 }
416 3
417 3               tloPtr->tlo->root.objLevel = RSTRPC_tlo_type;
418 3               tloPtr->tlo->root.objName = esl_strdup(wi_list->wi_name);
419 3               if (NULL == tloPtr->tlo->root.objName)
420 4               {
421 4                         result = EP_RB_RECOVER_NOMEM;
422 4                         break;
423 3               }
424 3
425 3               tloPtr->tlo->root.backupApp = 0;
                                                  /* value for network objects */
426 3
427 3               if (
                  NULL != wi_list->wi_work) /* may now be null for oldb kicker */
428 4                 {
429 4                   tloPtr->tlo->fileSpec = esl_strdup(wi_list->wi_work);
430 4                   if (NULL == tloPtr->tlo->fileSpec)
431 5                   {
432 5                         result = EP_RB_RECOVER_NOMEM;
433 5                         break;
434 4                   }
435 3                 }
436 3
437 3                 if (NULL != wi_list->wi_bic)
438 4                 {
439 4                   tloPtr->tlo->wiBIC = esl_strdup(wi_list->wi_bic);
440 4                   if (NULL == tloPtr->tlo->wiBIC)
441 5                   {
442 5                         result = EP_RB_RECOVER_NOMEM;
443 5                         break;
```

```
444 4                   }
445 3               }

447 3               tloPtr->tlo->wiType = wi_list->wi_type;

449 3               wi_list = wi_list->next;
450 2           }
451 2           else    /* get tlo from plugin list: */
452 3           {
453 3               if (NULL == *topLevObjs)
454 4               {   /* set start of list ptr to plugin tlo's */
455 4                   *topLevObjs = pluginList;
456 3               }
457 3               else
458 4               {   /* link prev entry to this one in plugin list */
459 4                   tloPtr->next = pluginList;
460 3               }
461 3               tloPtr = pluginList;
462 3               pluginList = pluginList->next;
463 3               tloPtr->next = NULL;
464 2           }


466 2           /* set hostname in all top level objects */
467 2           tloPtr->tlo->hostname = esl_strdup( sourceHost );
468 2           if (NULL == tloPtr->tlo->hostname)
469 3           {
470 3               result = EP_RB_RECOVER_NOMEM;
471 3               break;
472 2           }
473 2           ++total_count;
474 2           ++index;
475 1       }

477 1   /* catch all malloc failures here: free list, return failure (
                                                        nothing) */
476 2       if (result != E_SUCCESS) {
479 2           rec_api_log_csm(SUB_CSM_NOMEM, NULL);
480 2           index = 0;
                                            /* to indicate none returned */
481 2           valid_cookie = INIT_COOKIE;
                                        /* to indicate restart needed */
482 2           total_count = wi_count + pi_count + 1;
                                        /* to cause free of wi lists */
483 2           /* free linked list output */
484 2           if (*topLevObjs != NULL)
485 3           {
486 3                   RSTSL_FreeRestorableObjectList( *topLevObjs );
487 3                   *topLevObjs = NULL;
488 2           }
489 1       }

491 1       *numberEntries = index;

493 1       /*
494 1        * Set the cookie appropriately.
                                    Also store it in the static *valid_cookie*
495 1        */

497 1       if (wi_count + pi_count >= total_count)       /* any more TLO's for
                                                             later? */
498 2       {                                             /* yes: */
499 3           if (
            wi_count >= total_count) {              /*   any more netwk TLOs? */
500 3                   *cookie = (long)wi_list;
                                            /* yes, still have netwk WIs */
501 2           }
```



```
502 3           else {
503 3                   *cookie = (long)pluginList;
                                                    /* no, rest are plugin TLOs */
504 2           }
505 2           valid_cookie = *cookie;
506 1       }
507 1       else
508 2       {
509 2           /*
510 2            * Set the cookie to DONE state and the valid_cookie to match
                                                                        it.
511 2            */

513 2           if (result == E_SUCCESS)
514 3           { /*dont set cookie on errors */
515 3                   *cookie = DONE_COOKIE;
516 3                   valid_cookie = DONE_COOKIE;
517 2           }

519 2           /*
520 2            * Free up memory.  We are done and don't need it anymore.
521 2            * Start from the top of the list.
522 2            */

524 2           while (NULL != top_wi_list)
525 3           {
526 3               if (NULL != top_wi_list->wi_name)
527 4               {
528 4                   free(top_wi_list->wi_name);
529 3               }
530 3               if (NULL != top_wi_list->wi_work)
531 4               {
532 4                   free(top_wi_list->wi_work);
533 3               }
534 3               if (NULL != top_wi_list->wi_bic)
535 4               {
536 4                   free(top_wi_list->wi_bic);
537 3               }

539 3               tmp_list = top_wi_list->next;
540 3               free(top_wi_list);
541 3               top_wi_list = tmp_list;
542 2           }
543 2           if (pluginList)
544 3           {
545 3                   RSTSL_FreeRestorableObjectList( pluginList );
546 3                   pluginList = NULL;
547 2           }
548 1       }

550 1       return( result );
551   }       /* end of RSTSL_GetTopLevelObjects() */
```

D5

Fri Oct 10 14:42:35 2008 Function Index Page 2

Fri Oct 10 14:42:35 2008 Function Index Page 2

```
/*****************************************************************************
******

**
** File Name:   RSTgetrobs.c
**
** Copyright (c) 1998,1999 by EMC Corporation.
**
** Purpose:
**
** This module contains the EDMRST_GetRestorableObjects API and some
                                                                    of its
** support functions.
**
** Table of Contents:
** ------------------
**      API Functions:
**
**              EDMRST_GetRestorableObjects
**
**
** Compile-Time Options:
**
*****************************************************************************
                                                                    *****/



/* The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
*/

#ifndef lint
static char RCS_id [] = "$RCSfile$ "
                        "$Revision$ "
                        "$Date$" ;
#endif


/*
 * Feature test switches.
 * Standard defines required to turn on OS features go here.
 *
 * The following is required for code that uses POSIX API's.
 * Remove for non-POSIX, non-portable code.
 */

#define  _POSIX_SOURCE 1


/*
 * System headers.
 */
#include <unistd.h>       /* doesn't help define usleep(), so... */
extern int usleep( unsigned int );


/*
 * Epoch headers.
 */
#include <eb/eb_port.h>
#include <eb/rb_log.h>


/*
 * Local headers
 */
```



```
#include <RSTinterns.h>
#include <RSTsup_csm.h>


/*
 * #defines, structures, typedefs local to this source file
 */

#define RST_MAX_GET_ROBJS_DELAY 3        /* max seconds between polls
                                                                    */


/*
 * External declarations
 */

NEW_SRC_FILE();


/*
 * Local function prototypes
 */
```

```
85  /*****************************************************************
86   * EDMRST_GetRestorableObjects API
87   *
88   *    Function Description:
89   *
90   *  Given a parent (
91   *                  i.e. container) restorable object, return the children
     *  restorable objects it contains.
92   *
93   *  The cookie must be initialize to INIT_COOKIE on the first call to
                                                                    this
94   *  routine.
     *        This routine will update the cookie to allow retrieval of more
95   *  objects if there is more than *maxEntries*.  The cookie will be
96   *  returned as DONE_COOKIE when there are no more to retrieve.
97   *
98   *    Parameters:
99   *        svrHdl          - (
     *                        I) A pointer to this user's client handle for the
     *                           Restore Engine (server) connection.
100  *        parentPtr       - (I) ptr to parent restorableObject
101  *        allowBF         - (
     *                        I) flag indicating whether or not to include bad
     *                           files
103  *        maxEntries      - (
     *                        I) max. # of entries that the preallocated buffer
     *                           can hold
105  *        objBufPtr       - (
     *                        O) ptr to preallocated array of restorableObject
     *                           buffer ptrs
107  *        numEntries      - (
     *                        O) ptr to buffer to receive number of entries
     *                           returned in objBufArray
109  *        cookie          - (
     *                     I/O) ptr to a long integer whose value is meaningful
     *                           to only the internals of the API
111  *
     *                           == INIT_COOKIE: initial query input
112  *                         == DONE_COOKIE: end of query output
113  *
114  *  Return Codes:
115  *        E_SUCCESS       - operation completed successfully
116  *
117  *        EP_RB_RECOVER_INVAL_OBJNAME     - input restorableObject does
     *                                          not have a valid name;
118  *        EP_RB_RECOVER_BAD_ARGS          - input restorableObject ptr
     *                                          or objBufPtr is NULL;
119  *        EP_RB_RECOVER_BAD_COOKIE        - input cookie ptr is NULL or
     *                                          the cookie is DONE_COOKIE;
120  *        EP_RB_RECOVER_INVALOP           - the call is issued without
     *                                          the correct context setup;
121  *
122  *        return codes from GetWorkItemContents() and from
123  *        GetDirContents().
124  */
125
126 eerrno_ty
127 EDMRST_GetRestorableObjects( serverHandle        svrHdl,
128                              const restorableObjectPtr parentPtr,
129                               const boolean_ty    allowBF,
130                               const long                maxEntries,
131                              restorableObjectPtr *objBufPtr,
132                               long               *numEntries,
133                               long               *cookie )
134 {
135         RE_get_restorable_objects_start_result  *start_rpc_result =
```



```
                                                                    NULL;
142 1           RE_get_restorable_objects_start_args    start_rpc_args;
143 1           RE_get_restorable_objects_output_result
                                                  *output_rpc_result = NULL;
144 1           RE_get_restorable_objects_output_args   output_rpc_args;
145 1           RSTRPC_uro_list                         *temp_list;
146 1           struct RSTRPC_restorable_obj_root       *temp_robj;
147 1           eerrno_ty                          result = E_SUCCESS ;
148 1           short                              index;
149 1           restorableObject          **objPtrArray = (
                                                 restorableObject **)objBufPtr;


151 1           rbe_log_debug_sub( 0, "EDMRST_GetRestorableObjects called" );

153 1           /* validate args first: */
154 1           if (parentPtr==NULL
155 1             || numEntries==NULL
156 1             || cookie==NULL
157 1             || maxEntries <= 0
158 1             || objBufPtr== NULL
159 1             || svrHdl==NULL )
160 1                   return( EP_RB_RECOVER_BAD_ARGS );

162 1           /* validate target restorableObjects: */
163 1           for ( index=0; index<maxEntries; index++ )
164 2           {
165 2                   if ( (NULL == objPtrArray[index])
166 2                     || (
                      RESTORABLE_OBJECT != objPtrArray[index]->restoreObjType)
                          || (NULL != objPtrArray[index]->rpcObjPtr )
                        )
167 2                           return( EP_RB_RECOVER_BAD_ARGS );
168 2                   }
169 2           }
170 1

172 1           if (*cookie == DONE_COOKIE)
173 1                   return(EP_RB_RECOVER_BAD_COOKIE);

175 1           /* validate parent object type as top level or container */
176 1           if (NULL == (temp_robj = ((
                                  restorableObject *)parentPtr)->rpcObjPtr))
177 1                   return( EP_RB_RECOVER_BAD_ARGS );

179 1           if (RESTORABLE_OBJECT !=
180 1               ((restorableObject *)parentPtr)->restoreObjType )
181 1                   return EP_RB_RECOVER_INVAL_OBJTYPE;

183 1           if ( (temp_robj->objLevel != RSTRPC_tlo_type)
184 1             && (temp_robj->objLevel != RSTRPC_container_type) )
185 2           {
186 2                   if (temp_robj->objLevel != RSTRPC_leaf_type)
187 2                           return( EP_RB_RECOVER_INVAL_OBJTYPE );
188 2                   else
189 2                           return( EP_RB_RECOVER_INVALOP );
190 1           }

192 1           /* request the restorable objects with one RPC */

194 1           /* Prepare input argument structure for RPC: */
195 1           start_rpc_args.parentObj=(struct RE_restorable_obj) );

196 1           start_rpc_args.parentObj->objLevel = temp_robj->objLevel;
197 1           start_rpc_args.parentObj->RE_restorable_obj_u.uroInfo
198 1                   = (struct RSTRPC_user_restorable_object *)temp_robj;
199 1           start_rpc_args.allowBadFiles = allowBF;
200 1           start_rpc_args.cookie = *cookie;
201 1           start_rpc_args.maxEntries = maxEntries;
```

```
202 1           set_rpc_obj( re_get_restorable_objects_start,
203 1                        &start_rpc_args.RPCobjID );
205 1           start_rpc_result = re_get_restorable_objects_start_1(
                                                        &start_rpc_args,
206 1                                                       svrHdl );
207 2           if (!start_rpc_result) {
208 2                   result = EP_RB_RECOVER_RPC_FAIL;
209 2                   rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL);
210 1           }
211 1           else
212 2           {
213 2                   result = start_rpc_result->status;
215 2                   /* release RPC result struct: contents and struct */
216 2                   xdr_free( xdr_RB_get_restorable_objects_start_result,
217 2                             (char *)start_rpc_result );
218 1           }
220 1           free( start_rpc_args.parentObj );
222 1           /* prepare to call another RPC for results, if successful: */
223 1           if (result != E_SUCCESS)
224 1                   return( result );
225 1           else
226 1                   result = EP_RB_RECOVER_RPC_INCOMPLETE;
228 1           output_rpc_args.maxEntries = maxEntries;
230 1           /* poll for completion or error */
231 1           while (result == EP_RB_RECOVER_RPC_INCOMPLETE)
232 2           {
233 2                   unsigned int poll_delay = 100000;
                                                        /* .1 second */
234 2                   set_rpc_obj( re_get_restorable_objects_output,
235 2                                &output_rpc_args.RPCobjID );
236 2                   output_rpc_result =
                                re_get_restorable_objects_output_1(
237 2                                                      &output_rpc_args,
                                                              svrHdl );
238 2                   if (!output_rpc_result) {
239 3                           result = EP_RB_RECOVER_RPC_FAIL;
240 3                           rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL);
241 3                   }
242 2                   else
243 2                           result = output_rpc_result->status;
244 2
246 2                   if (result == EP_RB_RECOVER_RPC_INCOMPLETE)
247 3                   {
248 3                           /* release RPC result struct: contents and
                                                                struct */
249 3                           xdr_free(
                                    xdr_RB_get_restorable_objects_output_result,
                                            (char *)output_rpc_result );
250 3                           output_rpc_result = NULL;
251 3                           /* wait till next poll */
252 3                           usleep( poll_delay );
253 4                           if (poll_delay < RST_MAX_GET_ROBJS_DELAY) {
254 4                               poll_delay *= 2;
255 4                               if (poll_delay > RST_MAX_GET_ROBJS_DELAY)
256 4                                   poll_delay = RST_MAX_GET_ROBJS_DELAY;
257 3                           }
258 2                   }
259 1           }
260 1   }
```



```
262 1           /* move results to caller's area, if successful: */
263 1           if (result == E_SUCCESS)
264 2           {
265 2                   *cookie = output_rpc_result->cookie;
266 2                   *numEntries = output_rpc_result->numEntries;
267 2                   index = 0;
268 2                   while ( output_rpc_result->numEntries )
269 3                   {
270 3                           temp_list = output_rpc_result->childrenObjs;
271 3                           if (
                                    !temp_list || !output_rpc_args.maxEntries-- )
                                        break;
272 3                                   /* null pointer or too many returned */
273 3                           objPtrArray[index]->rpcObjPtr =
                                    (
                                     RSTRPC_restorable_obj_root *)temp_list->uro;
274 3                   /* needed to end with NULL in
                                                output_rpc_result->childrenObjs,
                     * because returned user rest.
                                                        objects can't be freed yet */
275 3                           output_rpc_result->childrenObjs =
                                                        temp_list->next;
276 3                           free( temp_list );
277 3                           output_rpc_result->numEntries--;
278 3                           index++;
279 3                   }
280 2                   if (output_rpc_result->numEntries)
281 2                           result = EP_RB_RECOVER_SERVERFAIL;
282 1           }
284 1           /* release RPC result struct's contents and itself: */
286 1           if (output_rpc_result) {
287 2                   xdr_free(
                                xdr_RB_get_restorable_objects_output_result,
288 2                              (char *)output_rpc_result );
289 2           }
290 1
292 1           return( result );
293     }     /* EDMRST_GetRestorableObjects */
```

Fri Oct 10 12:09:42 2008 Function Index Page 2

Fri Oct 10 12:09:42 2008 Function Index Page 1

Fri Oct 10 12:09:42 2008 Function Index Page 2

```
/*****************************************************************
                                                        ******/

**
** File Name:   RSLgetrobs.c
**
** Copyright (c) 1998,1999 by EMC Corporation.
**
** Purpose:
**
**       This module contains the RSTSL_GetRestorableObjects Restore
                                                        Service
**       Library API.
**
** Table of Contents:
** ------------------
**
**               RSTSL_GetRestorableObjects
**
**       Internal Functions:
**
**               GetTLOContents
**
** Compile-Time Options:
**               This section must list any compile time definitions
**               which will affect this header.
**
*****************************************************************
                                                        *****/


/* The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
*/

#ifndef lint
static char RCS_id [] = "$RCSfile$ "
                        "$Revision$ "
                        "$Date$" ;
#endif


/*
 * Feature test switches.
 * Standard defines required to turn on OS features go here.
 *
 * The following is required for code that uses POSIX API's.
 * Remove for non-POSIX, non-portable code.
 */

#define  _POSIX_SOURCE 1


/*
 * System headers.
 */


/*
 * Epoch headers.
 */
#include <eb/eb_port.h>
#include <eb/rb_log.h>
#include <ebutil/eb_locking.h>
```



```
/*
 * Local headers
 */
#include <RSLinterns.h>
#include <RSLstartup.h>


/*
 * #defines, structures, typedefs local to this source file
 */


/*
 * External declarations
 */

NEW_SRC_FILE();


/*
 * Local function prototypes
 */

static eerrno_ty GetTLOContents(
                                        struct RSTRPC_top_level_obj *tloObjPtr,
                                           const boolean_ty allowBF,
                                           const long maxEntries,
                                           struct RSTRPC_uro_list **objListPtr,
                                           long *cntEntries,
                                           long *cookie);
```

```
/*****************************************************************
 *  RSTSL_GetRestorableObjects:
 *
 *  This function is provided to allow retrieval of the
 *  child objects which are restorable given the parent object. The
 *  caller specifies the parent object and whether or not
 *  to include bad files.
 *
 *  Performs the 'get' on the asynchronous thread of the Restore
                                                        Engine.
 *
 * Parameters:
 *    parentPtr         (
                    I)  - the parent object: pointer to either a top level
                                  object or container object
 *    objectLevel       (
              I)  - specifies whether parentObject points to top level or
                                  container object
 *    objects           (
                   O)  - a pre-allocated pointer to return the list of
                                  objects in
 *    cookie          (IO)  - a place holder for the list position
 *    maxEntries      (I)   - the maximum number of objects to return
 *    cntEntries      (
                   O)  - the real number of objects returned in the list
 *    allowBF         (I)   - flag whether or not to include bad files
 *
 *****************************************************************/
eerrno_ty  RSTSL_GetRestorableObjects(
                                restorableObjectPtr       parentPtr,
                                        enum RSTRPC_ObjectLevel
                                                          objectLevel,
                                struct RSTRPC_uro_list
                                                          **objects,
                                long                      *cookie,
                                const long                maxEntries,
                                long                      *cntEntries,
                                const boolean_ty allowBF )
{
    eerrno_ty     result;
    struct RSTRPC_uro_list *uroList;
    struct RSTRPC_top_level_obj *tloPtr
          = (struct RSTRPC_top_level_obj *)parentPtr;
    struct RSTRPC_user_restorable_object *uroPtr
          = (struct RSTRPC_user_restorable_object *)parentPtr;

    /*
     *  Validate the input arguments.
     */

    if ((NULL == parentPtr) || (NULL == objects) || (
                                                    NULL == cntEntries) )
    {
        return(EP_RB_RECOVER_BAD_ARGS);
    }

    if (objectLevel == RSTRPC_tlo_type) {
        if (tloPtr->root.objName == NULL)
        {
                return(EP_RB_RECOVER_INVAL_OBJNAME);
        }
    }
    else if (objectLevel == RSTRPC_container_type) {
        if (uroPtr->root.objName == NULL)
        {
                return(EP_RB_RECOVER_INVAL_OBJNAME);
```



```
        }
        if (rcp->rc_backup_app != uroPtr->root.backupApp)
        {   /* input object not from same app as current TLO */
                return EP_RB_RECOVER_INVALOP;
        }
    }
    else if (objectLevel == RSTRPC_leaf_type) {
        return(EP_RB_RECOVER_INVALOP);
    }
    else {
        return(EP_RB_RECOVER_INVAL_OBJTYPE);
    }

    /*
     * If cookie is NULL of if it indicates DONE_COOKIE, return error.
     */

    if ((cookie == NULL) || (*cookie == DONE_COOKIE))
    {
        return(EP_RB_RECOVER_BAD_COOKIE);
    }

    /*
     * If EDMRST_Initialize() has not been called, or for some reason
     * the recover_context has not been allocated, then bail out
     * right away!
     */

    if (rcp == NULL)
    {
        return(EP_RB_RECOVER_INVALOP);
    }

    /*
     * If EDMRST_GetTopLevelObjects has not been called,
                                                    then this operation
     * is invalid.
     */

    if (rcp->rc_source_client_hostname == NULL)
    {
        return(EP_RB_RECOVER_INVALOP);
    }

    /*
     * Set number of entries returned to be 0 in case we encounter an
     * error before setting it appropriately.
     */

    *cntEntries = 0;

    /*
     *  If the parent object is a top level object,
                                                call GetTLOContents();
     *  if parent object is a container (
                                        directory), call RSTLL_GetDirContents
     *  for network backups,      and RSTPI_GetNextLevelObjects for other backup
     *  apps;
              else return error indicating parent object is not a parent.
     */

    if (objectLevel == RSTRPC_tlo_type) {
        result = GetTLOContents( tloPtr,
                                 allowBF,
                                 maxEntries,
```

```
                                        objects,
                                        cntEntries,
                                        cookie );
    }
    else        /* we already know its a container */
    {
        /*
         * Top Level Object must have been established before this
                                                                    func
         * is called to list dir contents.
         */
        if (rcp->rc_top_level_object_name == NULL)
        {
            return EP_RB_RECOVER_INVALOP;
        }

        if (NULL != rcp->currentPIptr)
        {
                result =
                    rcp->currentPIptr->piFuncArray[PIFuncIndexGetNLO]
                                                    ( rcp,
                                                      uroPtr,

                                                    RSTRPC_container_type,
                                                      objects,
                                                      cookie,
                                                      maxEntries,
                                                      cntEntries,
                                                      allowBF );
        }
        else
        {
                result = RSTLL_GetDirContents( rcp,
                                                    uroPtr->root.objName,
                                                    allowBF,
                                                    maxEntries,
                                                    objects,
                                                    cntEntries,
                                                    cookie );
        }
    }

    /* if successful, mark all objects with proper backup app: */
    if (E_SUCCESS == result)
    {
        for ( uroList = *objects; uroList; uroList = uroList->next )
                uroList->uro->root.backupApp = rcp->rc_backup_app;
    }

    return result;

}       /* RSTSL_GetRestorableObjects */
```



```
/*
 * GetTLOContents()
 *
 * Function Description:
 *      Given the name of the top level object,
                                                    set up the restore_context
 *      for it.
 *
 * Parameters:
 *      tloObjPtr   - (I) ptr to the work item restorableObject
 *      allowBF             - (
                        I) flag indicating whether or not to include bad files
 *      maxEntries  - (
                        I) max. # of entries that the preallocated buffer can
                                                                    hold
 *      objListPtr  - (
                        O)  - a pre-allocated pointer to return the list of
                                                                objects in
 *      cntEntries  - (
                    O) ptr to buffer to receive number of entries returned
                                                                in objBufPtr
 *      cookie      - (
                    I/O) ptr to a long integer whose value is meaningful to
                                                                only the internals of the API
 *
 * Return code:
 *      E_SUCCESS           - success. (It is defined as 0 in e_errno.h)
 */

static eerrno_ty
GetTLOContents(  struct RSTRPC_top_level_obj *tloPtr,
                 const boolean_ty allowBF,
                 const long maxEntries,
                 struct RSTRPC_uro_list **objListPtr,
                 long *cntEntries,
                 long *cookie)
{
    char *tloNameP = tloPtr->root.objName;
    char *tNameP = tloPtr->templateName;
    char *rcTmpNameP;
    eerrno_ty rc;
    boolean_ty reset = FALSE;
    int  index;


    /*
     * The environment needs to be reset under the following
     * situations:
     * - it's the first time ever that a top level object is selected;
     * - the caller wants to switch to a top level object different
                                                                    from
     *   what the env is set up for currently;
     *
     * The following criteria are used to determine if the caller
     * wants to switch to a different top level object:
     * - The backup application in the restorableObject is
     *   different from what's kept in the restore_context;
     * - the object name specified in the restorableObject is
     *   different from what's kept in the restore_context;
     * - the object name in the restorableObject is the same
     *   as what's in the restore_context, but the template or
     *   the trailset used is different,
                                                    or the client host is different;
     */
```

```
320 1     if ( (0 != strcmp(
                    tloPtr->hostname, rcp->rc_source_client_hostname ))
321 1       || (rcp->rc_backup_app != tloPtr->root.backupApp) )
322 2     {
323 2         reset = TRUE;   /* switch to a different application! */
324 1     }

326 1     else if ( (rcp->rc_top_level_object_name == NULL)
327 1          || (0 != strcmp(rcp->rc_top_level_object_name, tloNameP)))
328 2     {
329 2         reset = TRUE;   /* switch to a different object */
330 1     }
331 1     else
332 2     {
333 2          /*
334 2           * rc_top_level_object_name is already set and it's the same
335 2           * as the input top level obj name, we need to further check
336 2           * the template. If the template is the same, we must
337 2           * then check the trailset usage. If any of these is
338 2           * different between what's in the input restorableObject
339 2           * and what's in the restore_context, we need to reset
340 2           * the environment.
341 2           */

343 2          if (rcp->rc_template_defaulted)
344 2               rcTmpNameP = NULL;
345 2          else
346 2               rcTmpNameP = rcp->rc_template_name;

348 2          if (NULL != tNameP)
349 3          {
350 3              if (NULL == rcTmpNameP)
351 4              {
352 4                  reset = TRUE;
353 3              }
354 3              else if (strcmp(tNameP, rcTmpNameP)) /* diff templ */
355 4              {
356 4                  reset = TRUE;
357 3              }
358 3              else if (rcp->rc_saveset_thread != tloPtr->asThread)
359 4              {
360 4                  reset = TRUE;
361 3              }
362 2          }
363 2          else if (NULL != rcTmpNameP)
364 3          {
365 3              reset = TRUE;
366 2          }
367 1     }

369 1     if (reset)
370 2     {
371 2          if (*cookie != INIT_COOKIE)
372 3          {
373 3              return(EP_RB_RECOVER_INVALOP);
374 2          }

376 2     /* if last tlo was other than a network backup object,
                                                  let app clean up: */
377 2          if (rcp->rc_top_level_object_name != NULL)
378 3          {
379 3              if (NULL != rcp->currentPIptr)
380 4              {
381 4                  rc =
                        rcp->currentPIptr->piFuncArray[PIFuncIndexClearRC]( rcp );
```



```
382 4                   if (rc != E_SUCCESS)
383 5                   {
384 5                       rbc_internal_error( rc,
385 5                                  "plugin: %s error in
                                                  ClearRestoreContext call",
386 5                                  ( (struct pluginIDData *)(
                                            rcp->currentPIptr->idData) )->name );
387 4                   }
388 4                   rcp->currentPIptr->appData = rcp->appData;
                                                  /* save its data */
389 3               }
390 3               else
391 4               {
392 4               /* for network backups, just clear the mark context: */
393 4                   reset_marks( rcp );
394 3               }

396 3               free( rcp->rc_top_level_object_name );
397 3               rcp->rc_top_level_object_name = NULL;
398 2           }

400 2           /*
401 2            * If we're switching to a new source host, we must free
402 2            * the space used for the previous source host name string
                                                              before
403 2            * resetting it to the new.
404 2            */

406 2           if (0 != strcmp(
                        tloPtr->hostname, rcp->rc_source_client_hostname))

407 3           {
408 3               if (NULL != rcp->rc_source_client_hostname)
409 4               {
410 4                   free(rcp->rc_source_client_hostname);
411 3               }
412 3               rcp->rc_source_client_hostname = esl_strdup(
                                                  tloPtr->hostname );
413 3               if (NULL == rcp->rc_source_client_hostname)
414 4               {
415 4                   rec_api_log_csm(SUB_CSM_NOMEM, NULL);
416 4                   return(EP_RB_RECOVER_NOMEM);
417 3               }
418 2           }

420 2           if (0 == (rcp->rc_backup_app = tloPtr->root.backupApp))
421 3           { /* new app is network: */
422 3                   rcp->appData = NULL;
423 3                   rcp->currentPIptr = NULL;
424 2           }
425 2           else  /* find app in plugin list,
                                  set its appData and pluginData ptrs */

426 3           {
427 3                   for (index=1, rcp->currentPIptr = rcp->piList;
428 3                        index < rcp->rc_backup_app &&
                                                  rcp->currentPIptr->next ;

429 3                        index++
430 3                       )
431 3                           rcp->currentPIptr = rcp->currentPIptr->next;
432 3                   if (index != rcp->rc_backup_app)
433 4                   { /* got null ptr too early */
434 4                       rbc_internal_error( EP_RB_RECOVER_FATALERR,
435 4                                  "plug-in list corrupted"
                                                              );
436 4                       rec_api_log_csm( SUB_CSM_INTERNAL_ERR, NULL );
437 4                       return EP_RB_RECOVER_FATALERR;
438 3                   }
```

```
440  3                  rcp->appData =  rcp->currentPIptr->appData;
441  2              }

443  2          /*
444  2           * If we're switching to a different tlo, we must:
445  2           * unlock any previous work items and free the workitem name
                                                                    string
446  2           */
447  2          if (rcp->rc_workitem_name != NULL)
448  3          {
449  3              if (rcp->rc_have_wilock)
450  4              {
451  4                  eb_unlock_object(
                                EB_OBJECT_WORKITEM, rcp->rc_workitem_name,
                                    EB_UNLOCK_FREE_IT);
452  4                  rcp->rc_have_wilock = 0;
453  3              }
454  3              free(rcp->rc_workitem_name);
455  3              rcp->rc_workitem_name = NULL;
456  2          }

459  2          if (rcp->rc_template_name != NULL)
460  3          {
461  3              free(rcp->rc_template_name);
462  2          }

464  2          if (tNameP != NULL)
465  3          {
466  3              rcp->rc_template_name = esl_strdup( tNameP );
467  3              rcp->rc_template_defaulted = FALSE;
468  3              if (NULL == rcp->rc_template_name)
469  4              {
470  4                  rec_api_log_csm(SUB_CSM_NOMEM, NULL);
471  4                  return(EP_RB_RECOVER_NOMEM);
472  3              }
473  2          }
474  2          else
475  3          {
476  3              rcp->rc_template_name = NULL;
477  3              rcp->rc_template_defaulted = TRUE;
478  2          }

480  2          rcp->rc_saveset_thread = tloPtr->ssThread;

482  2          /* Do this last,
                            to make sure all strdup's succeed: else leave
483  2           * rc_top_level_object_name NULL */
484  2          rcp->rc_top_level_object_name = esl_strdup(tloNameP);
485  2          if (NULL == rcp->rc_top_level_object_name)
486  3          {
487  3              rec_api_log_csm(SUB_CSM_NOMEM, NULL);
488  3              return(EP_RB_RECOVER_NOMEM);
489  2          }

491  2          /*
492  2           * Check if restore is authorized to the specified client
                                                                    system(s)
493  2           * and by the specified user.
494  2           */

496  2          check_source_sys_admin(rcp);

498  2          if (0 != rcp->rc_backup_app)
499  3          {
500  3                  rc =
```



```
                        rcp->currentPIptr->piFuncArray[PIFuncIndexSetTLO]
                                                                    (
501  3                                                                  rcp, tloPtr);

502  2              }
503  2              else
504  3              {
505  3              /* for network backups, init the workitem & mark context: */
506  3                      rc = RSTLL_SetWorkItem( rcp, tloNameP );
507  3                      if (rc == E_SUCCESS)
508  3                              rc = alloc_plane_arrays( rcp );
509  2              }

511  2              if (rc != E_SUCCESS)
512  3              {
513  3                  free( rcp->rc_top_level_object_name );
514  3                  rcp->rc_top_level_object_name = NULL;
515  3                  return(rc);
516  2              }
517  1      }

519  1      if (0 != rcp->rc_backup_app)
520  2      {
521  2          return rcp -> currentPIptr -> piFuncArray[PIFuncIndexGetNLO]
                                                                    ( rcp,
522  2                                                                  tloPtr,

523  2
524  2                                                              RSTRPC_tlo_type,
                                                                        objListPtr,
525  2                                                                  cookie,
526  2                                                                  maxEntries,
527  2                                                                  cntEntries,
528  2                                                                  allowBF );
529  2
530  1      }
531  1      else
532  2      {
533  2      /* for network backups retrieve content for the root directory.
                                                                    */

535  2          return RSTLL_GetDirContents( rcp,
536  2                                       "/",
537  2                                       allowBF,
538  2                                       maxEntries,
539  2                                       objListPtr,
540  2                                       cntEntries,
541  2                                       cookie );
542  1      }
543  }       /* GetTLOContents */
```

D6

```
/*****************************************************************
                                                          ******

**
** File Name:    RSTmarkunm.c
**
** Copyright (c) 1998,1999 by EMC Corporation.
**
** Purpose:
**              This module contains the Restore API functions to
                                                          mark and
**              unmark objects for restoral.
**
** Table of Contents:
** ------------------
**      API Functions:
**              EDMRST_MarkObject
**              EDMRST_GetMarkResults
**              EDMRST_UnmarkObject
**              EDMRST_GetUnmarkResults
**              EDMRST_GetMarkedTotalSize
**
**      Internal Functions:
**
** Compile-Time Options:
**
**
** NOTE: Part of this module is adapted from:
** server/libs/recover/grandfathered/cmd_markunmark.c
** It contains mainly support routines needed by the mark and unmark
** API functions.
**
*****************************************************************
                                                          ******/


/* The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
*/

#ifndef lint
static char RCS_id [] = "$RCSfile$ "
                        "$Revision$ "
                        "$Date$" ;
#endif


/*
 * Feature test switches.
 * Standard defines required to turn on OS features go here.
 *
 * The following is required for code that uses POSIX API's.
 * Remove for non-POSIX, non-portable code.
 */

#define  _POSIX_SOURCE 1


/*
 * System headers.
 */


/*
 * Epoch headers.
 */
```



```
#include <eb/eb_port.h>
#include <eb/rb_log.h>


/*
 * Local headers
 */
#include <RSTinterns.h>
#include <RSTsup_csm.h>


/*
 * #defines, structures, typedefs local to this source file
 */


/*
 * External declarations
 */


NEW_SRC_FILE();

/*
 * Local function prototypes
 */


/*****************************************************************
 * EDMRST_MarkObject()
 *   This function is passed a restorableObject and begins to mark
                                                          files for
 *   restoral based on the input criteria.  A second function,
 *   EDMRST_GetMarkResults,
                                  is used to test for completion of the mark.
 *
 * Parameters:
 *
 *   svrHdl      (I) - A pointer to this user's client handle for the
 *                     Restore Engine (server) connection.
 *   thisObject (I) - The input object to mark, must be of type file or
 *                    directory (not a top level object) .
 *     1) For files thisObject is the represents the target file to be
                                                          marked.
 *     2) For directories thisObject represents the directory to be
                                                          recovered
 *     and if the descend parameter is true then the mark applies to
                                                          all the
 *     contents of the directory.
 *     3) For Witems thisObject an error condition will be returned.
 *
 *   time        (I) - (
                       optional) the backup time to perform the mark on --
 *                      if not specified, uses currrently selected backup;
                                                          if
 *                      specified, leaves selected backup time unchanged
 *   allowBadfiles (I) - allows marking of files of state BADDATA.
 *   descend     (
                  I) - Should mark operation descend to operate on the contents
 *                      of directories.
 *
 *****************************************************************/

errno_ty
EDMRST_MarkObject( serverHandle      svrHdl,
                   restorableObjectPtr   thisObject,
```

```
                        time_t                  time,
                        boolean_ty              allowBadFiles,
                        boolean_ty              descend )
{
        RE_mark_object_result           *rpc_result;
        RE_mark_object_args             rpc_args;
        RSTRPC_restorable_obj_root      *temp_robj;
        eerrno_ty                       result = S_SUCCESS ;

        rbe_log_debug_sub( 0, "EDMRST_MarkObject called" );

        /* validate args first: */
        if (thisObject==NULL || svrHdl==NULL )
                return( EP_RB_RECOVER_BAD_ARGS );

        /* validate input object type as RESTORABLE_OBJECT */
        temp_robj = ((restorableObject *)thisObject)->rpcObjPtr;
        if ( NULL == temp_robj
          || RESTORABLE_OBJECT !=
                        (( restorableObject *)thisObject)->restoreObjType )
                return EP_RB_RECOVER_INVAL_OBJTYPE;

        /* validate input object type as NOT top level */
        if ( (temp_robj->objLevel != RSTRPC_leaf_type)
          && (temp_robj->objLevel != RSTRPC_container_type) )
        {
                if (temp_robj->objLevel != RSTRPC_tlo_type)
                        return( EP_RB_RECOVER_INVAL_OBJTYPE );
                else
                        return( EP_RB_RECOVER_INVALOP );
        }

        /* Prepare input argument structure for RPC: */
        rpc_args.thisObj = (RSTRPC_user_restorable_object *)temp_robj;
        rpc_args.allowBadFiles = allowBadFiles;
        rpc_args.descend = descend;
        rpc_args.backupTime = time;
        set_rpc_obj( re_mark_object, &rpc_args.RPCobjID );

        rpc_result = re_mark_object_1( &rpc_args, svrHdl );

        if (!rpc_result) {
                result = EP_RB_RECOVER_RPC_FAIL;
                rec_epi_log_csm( SUB_CSM_RPC_FAIL, NULL);
        }
        else {
                result = rpc_result->status;
                /* release RPC result struct: */
                xdr_free( xdr_RE_mark_object_result, (
                                                char *)rpc_result);
        }

        return( result );

}       /* end of EDMRST_MarkObject () */
```

```
177  /*********************************************************************
178   * EDMRST_GetMarkResults()
179   *
180   *       This function tests for completion and retrieves the results
                                                                    of the
181   * previously started mark operation.
182   *
183   * Parameters:
184   *
185   *  svrHdl      (I) - A pointer to this user's client handle for the
186   *                    Restore Engine (server) connection.
187   *
188   *  interrupt   (I) - requests cancellation of the mark (if TRUE)
189   *                    WARNING: If the operation is aborted,
                                                        the mark will be
190   *                    left in an unknown state.  That is,
                                                    any one of the
191   *                    descendants of the marked object may be marked
                                                            or not.
192   *                    It is up to the caller to determine how to
                                                            proceed
193   *                    afterwards.
194   *  BadFilesCount (
          O) -- returns the file count of bitfiles marked with BADDATA
195   *  PermDenyFilesCount (
                O) -- returns the file count with permission denied
196   *              bitfiles that were not marked.
197   *
198   *  fileMarked (
            O) -- return the total files marked after this mark occurred.
199   *  dirMarked (
            O) -- return the total directories marked after this mark
200   *              occurred.
201   *  otherMarked (
            O) -- return the total "other" files marked after this mark.
202   *
203   *********************************************************************/

205  eerrno_ty
206  EDMRST_GetMarkResults(   serverHandle     svrHdl,
207                           boolean_ty       interrupt,
208                            u_long           *BadFilesCount,
209                            u_long           *PermDenyFilesCount,
210                            u_long           *fileMarked,
211                            u_long           *dirMarked,
212                            u_long           *otherMarked )
213  1 {
214  1       RE_get_mark_results_result      *rpc_result;
215  1       RE_get_mark_results_args        rpc_args;
216  1       eerrno_ty                       result = E_SUCCESS ;

218  1       rbe_log_debug_sub( 0, "EDMRST_GetMarkResults called" );

220  1       /* validate args first: */
221  1       if ( svrHdl==NULL || BadFilesCount==NULL
222  1          || fileMarked==NULL || PermDenyFilesCount==NULL
223  1          || dirMarked==NULL || otherMarked==NULL
224  1       )
225  1               return( EP_RB_RECOVER_BAD_ARGS );

227  1       rpc_args.interrupt = interrupt;
228  1       set_rpc_obj( re_get_mark_results, &rpc_args.RPCobjID );
229  1       rpc_result = re_get_mark_results_1( &rpc_args, svrHdl );

231  2       if (!rpc_result) {
232  2               result = EP_RB_RECOVER_RPC_FAIL;
```



```
233  2               rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL);
234  1       }
235  2       else {  /* move results to caller's area, if successful: */
236  2               result = rpc_result->status;
237  2               if (result == E_SUCCESS)
238  3               {
239  3                       *BadFilesCount = rpc_result->badFileCount;
240  3                       *PermDenyFilesCount =
                                        rpc_result->permDenyFileCount;
241  3                       *dirMarked = rpc_result->dirMarkCount;
242  3                       *fileMarked = rpc_result->fileMarkCount;
243  3                       *otherMarked = rpc_result->otherMarkCount;
244  2               }

246  2               /* release RPC result struct: */
247  2               xdr_free( xdr_RE_get_mark_results_result, (
                                                        char *)rpc_result );
248  1       }

250  1       return( result );
251  }
```

```
253  /*****************************************************************
256   * UnmarkObject() and GetUnmarkResults()
255   *
256   *   UnmarkObject operates like MarkObject,
                                        in that it is supported through
257   *   two API calls -- UnmarkObject and GetUnmarkresults.
                                                    Unmark starts an
258   *   asynchronous operation in the Restore Engine to perform the
                                                          unmarking,
259   *   and returns.
260   *
261   *   GetUnmarkResults is called to test for completion of the unmark
                                                          operation,
262   *   and receive results when it is done.
                                          It can also be used to interrupt
263   *   the unmark operation.
264   *
265   * UnmarkObject Parameters:
266   *
267   *  svrHdl        (I) - A pointer to this user's client handle for the
268   *                      Restore Engine (server) connection.
269   *  thisObject    (I) - The restoral object;
                                              can be a leaf object (e.g. a
270   *                     file), or a container object (          e.g., a directory).
271   *  backupTime    (I) - (
                      optional) the backup time to perform the unmark on --
272   *                     if not specified,
                                    uses currrently selected backup; if
273   *                     specified,
                                  leaves selected backup time unchanged
274   *  BadFilesOnly  (
                      I) - allows unmarking ONLY of files of state BADDATA.
275   *  descend       (
                     I) - Should unmark operation descend to operate on the
276   *                     content of container objects.
277   *****************************************************************/

279  eerrno_ty EDMRST_UnmarkObject( serverHandle       svrHdl,
280                                 restorableObjectPtr thisObject,
281                                 time_t              backupTime,
282                                 boolean_ty          BadFilesOnly,
283                                 boolean_ty          descend )
284 1 {
285 1       RE_mark_object_result          *rpc_result;
286 1       RE_unmark_object_args          rpc_args;
287 1       RSTRPC_restorable_obj_root     *temp_robj;
288 1       eerrno_ty                      result = E_SUCCESS ;

290 1       rbe_log_debug_sub( 0, "EDMRST_UnmarkObject called" );

292 1       /* validate args first: */
293 1       if (thisObject==NULL || svrHdl==NULL )
294 1               return( EP_RB_RECOVER_BAD_ARGS );

296 1       /* validate input object type as RESTORABLE_OBJECT */
297 1       temp_robj = ((restorableObject *)thisObject)->rpcObjPtr;
298 1       if (NULL == temp_robj || RESTORABLE_OBJECT !=
299 1                               ((
                        restorableObject *)thisObject)->restoreObjType )
300 1               return EP_RB_RECOVER_INVAL_OBJTYPE;

302 1       /* validate input object type as NOT top level */
303 1       if ( (temp_robj->objLevel != RSTRPC_leaf_type)
304 1            && (temp_robj->objLevel != RSTRPC_container_type) )
```



```
305 2           {
306 2                   if (temp_robj->objLevel != RSTRPC_tlo_type)
307 2                           return( EP_RB_RECOVER_INVAL_OBJTYPE );
308 2                   else
309 2                           return( EP_RB_RECOVER_INVALOP );
310 1           }

312 1           /* Prepare input argument structure for RPC: */
313 1           rpc_args.thisObj = (RSTRPC_user_restorable_object *)temp_robj;
314 1           rpc_args.badFilesOnly = BadFilesOnly;
315 1           rpc_args.descend = descend;
316 1           rpc_args.backupTime = backupTime;
317 1           set_rpc_obj( re_unmark_object, &rpc_args.RPCobjID );

319 1           rpc_result = re_unmark_object_1( &rpc_args, svrHdl );

321 2           if (!rpc_result) {
322 2                   result = EP_RB_RECOVER_RPC_FAIL;
323 2                   rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL);
324 1           }
325 2           else {
326 2                   result = rpc_result->status;
327 2                   /* release RPC result struct: */
328 2                   xdr_free( xdr_RE_mark_object_result, (
                                                        char *)rpc_result);
329 1           }

331 1           return( result );
332   }
```

```
333   /* end of EDMRST_UnmarkObject () */

335   /*********************************************************************
336    *
337    * GetUnmarkResults Parameters:
338    *
339    *  svrHdl         (I) - A pointer to this user's client handle for the
340    *                       Restore Engine (server) connection.
341    *  interrupt      (I) - requests cancellation of the unmark (if TRUE)
342    *                       WARNING: If the operation is aborted,
                                                        the unmark will
343    *                       be left in an unknown state.  That is,
                                                        any one of the
344    *                       descendants of the unmarked object may be
                                                        marked or
345    *                       not.
                                    It is up to the caller to determine how to
346    *                       proceed afterwards.
347    *  BadFilesCount (O) - returns the file count with BADDATA.
348    *  fileMarked    (
            O) - return the total files marked after this mark occurred.
349    *  dirMarked     (
                  O) - return the total directories marked after this mark
350    *                       occurred.
351    *  otherMarked   (
              O) - return the total "other" files marked after this mark.
352    *
353    *********************************************************************/

355   eerrno_ty EDMRST_GetUnmarkResults( serverHandle svrHdl,
356                                      boolean_ty   interrupt,
357                                      u_long       *BadFilesCount,
358                                      u_long       *fileMarked,
359                                      u_long       *dirMarked,
360                                      u_long       *otherMarked )
361 1 {
362 1       RE_get_unmark_results_result    *rpc_result;
363 1       RE_get_mark_results_args        rpc_args;
364 1       eerrno_ty                       result = E_SUCCESS ;

366 1       rbe_log_debug_sub( 0, "EDMRST_GetUnmarkResults called" );

368 1       /* validate args first: */
369 1       if ( svrHdl==NULL || BadFilesCount==NULL || fileMarked==NULL
370 1         || dirMarked==NULL || otherMarked==NULL
371 1         )
372 1               return( EP_RB_RECOVER_BAD_ARGS );

374 1       rpc_args.interrupt = interrupt;
375 1       set_rpc_obj( re_get_unmark_results, &rpc_args.RPCobjID );
376 1       rpc_result = re_get_unmark_results_1( &rpc_args, svrHdl );

378 2       if (!rpc_result) {
379 2             result = EP_RB_RECOVER_RPC_FAIL;
380 2            rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL);
381 1       }
382 2       else {   /* move results to caller's area, if successful: */
383 2              result = rpc_result->status;
384 2              if (result == E_SUCCESS)
385 3              {
386 3                      *BadFilesCount = rpc_result->badFileCount;
387 3                      *dirMarked = rpc_result->dirMarkCount;
388 3                      *fileMarked = rpc_result->fileMarkCount;
389 3                      *otherMarked = rpc_result->otherMarkCount;
390 2              }
```



```
392 2                   /* release RPC result struct: */
393 2                   xdr_free( xdr_RE_get_unmark_results_result, (
                                                        char *)rpc_result );
394 1           }

396 1           return( result );
397     }
```

```
399  /*********************************************
400   * EDMRST_GetMarkedTotalSize ()
401   *
402   * This function is provided to allow retrieval of the
403   * Total size of the marked files.
404   *
405   * size is the sum-of-the-length the marked files and is one
406   * measure of size. This is an approximation.
407   *
408   *********************************************/
409
410  eerrno_ty
411  EDMRST_GetMarkedTotalSize( serverHandle svrHdl,
412                             u_hyper       *totalSize )
413  1 {
414  1       RE_get_marked_total_size_result         *rpc_result;
415  1       RE_null_args                            rpc_args;
416  1       eerrno_ty                               result = E_SUCCESS ;
417
418  1       rbe_log_debug_sub( 0, "EDMRST_GetMarkedTotalSize called" );
419
420  1       /* validate args first: */
421  1       if ( svrHdl==NULL || totalSize==NULL )
422  1               return( EP_RB_RECOVER_BAD_ARGS );
423
424  1       set_rpc_obj( re_get_marked_total_size, &rpc_args.RPCobjID );
425  1       rpc_result = re_get_marked_total_size_1( &rpc_args, svrHdl );
426
427  2       if (!rpc_result) {
428  2               result = EP_RB_RECOVER_RPC_FAIL;
429  2              rec_api_log_csm( SUB_CSM_RPC_FAIL, NULL);
430  1       }
431  2       else {   /* move results to caller's area, if successful: */
432  2               result = rpc_result->status;
433  2               if (result == E_SUCCESS)
434  3               {
435  3                       totalSize->high = rpc_result->total.high;
436  3                       totalSize->low = rpc_result->total.low;
437  2               }
438
439  2               /* release RPC result struct: */
440  2               xdr_free( xdr_RE_get_marked_total_size_result,
441  2                       (char *)rpc_result );
442  1       }
443
444  1       return( result );
445  }
```



```
446  /*      EDMRST_GetMarkedTotalSize */
```